HUMBOLDT-UNIVERSITÄT ZU BERLIN

Design und Management von Experimentier-Workflows

Dissertation

zur Erlangung des akademischen Grades
doctor rerum naturalium (Dr. rer. nat.)
im Fach Informatik

eingereicht an der
Mathematisch-Naturwissenschaftlichen Fakultät II
der Humboldt-Universität zu Berlin

von
Diplom-Informatiker Frank Kühnlenz

Präsident der Humboldt-Universität zu Berlin
Prof. Dr. Jan-Hendrik Olbertz

Dekan der Mathematisch-Naturwissenschaftlichen Fakultät II
Prof. Dr. Elmar Kulke

Gutachter:

1. Prof. Dr. Joachim Fischer
2. Prof. Dr. Holger Schlingloff
3. Prof. Dr. Bertram Ludäscher

Tag der Verteidigung: 18.11.2014

Abstract

Experimentation in my work means performing experiments based on computer-based models, which describe system structure and behaviour abstractly. Often, these systems do not exist in reality yet or can not be explored directly for other reasons. Hence, instead of the system itself models of the system will be explored (with a certain goal). These models are executed in special computing environments (usual simulation systems) to get model observations. Systematic experimentation using model input parameter variation assignments leads to lots of possibly long-running experiments that must be planned, documented, automated executed, monitored and evaluated.

The problem is, that experimenters (who are usually not computer scientists) miss the proper means of expressions (e. g., to express variations of parameter assignments) to describe experimentation processes formally in a way, that allows their automatic execution by a computer system while preserving reproducibility, re-usability and comprehension (summarized as *transparency*).

An existing approach to tackle this problem is using a *scientific workflow management system (S-WfMS)*, which is applicable, since the process of experimentation can be defined as a specifically structured scientific process that comprises the ordered phases planning, execution and evaluation of experiments. A formalization of such a process is further called as *experimentation workflow* (a specialization of a scientific workflow). A S-WfMS is flexible enough to handle experimentation workflows, while using *only* the concepts of data and control flow. It contains no knowledge about the semantics of the data and how it is related to the model or the experimentation aim. This is problematic, because experimenters are focused on the aim.

My approach is to identify general experimentation workflow concepts and formalize them as a meta-model-based *domain-specific language (DSL)* that I call *experimentation language (*ExpL*)*. *experimentation language (*ExpL*)* includes general workflow concepts like control flow and the composition of activities, and it allows modeling of experimentation workflows on a framework-independent, conceptional level. This approach is based on the idea to apply the paradigm of *model driven engineering* also to the domain of experimentation workflows. Hence, re-using and sharing the experimentation workflow with other scientists is not limited to a particular framework anymore. Therefore, it is easier to switch the particular framework that executes the model because the experimentation description does not need to be re-implemented. This is useful in order to follow different experimentation aims if the type of the model supports switching the framework.

ExpL includes common workflow concepts like activities and resources, but provides declarative elements in order to support parameter studies properly, too. This is a novelty that helps

to provide a proper level of abstraction to the experimenter. ExpL is always being used in a specific experimentation domain (e. g., within the field of geography or physics) that has certain specifics in configuration and evaluation methods. Addressing this, I propose to separate the concerns and use two other, dependent *domain-specific languages (DSLs)* additionally for *configuration* and *evaluation*. For example, such a *configuration DSL* realizes concepts for data structures, operators and units. An *evaluation DSL* should provide means to formulate metric functions and the application of optimization methods.

Using ExpL offers several advantages compared to existing approaches: tailored means of expressions in the experimenter's vocabulary make it easier to use compared to S-WfMSs. The vocabulary of ExpL covers the experimentation workflow with a consistent documentation and clear provenance of all its artifacts completely. This leads to an improved understanding of the experiments that can be published with lesser effort to other scientists. They can reproduce the results, even if they are not computer scientists.

Three interdisciplinary case studies for formalizing experimentation processes from three different domains are shown. These experimentation processes deal with models of the following types: a land-use change model (SLEUTH), a model that represents a sensor network for earthquake early warning (SOSEWIN) and a model that describes optical nanostructures.

Zusammenfassung

Experimentieren in der vorliegenden Arbeit bedeutet, Experimente auf der Basis von computerbasierten Modellen durchzuführen, wobei diese Modelle Struktur und Verhalten eines Systems abstrahiert beschreiben. Oftmals sind die dabei betrachteten Systeme entweder (noch) nicht physisch existent oder durch anderweitige Gründe einer realen Untersuchung schwer zugänglich. Deshalb untersucht man stellvertretend für das System ein Modell dieses Systems (das speziell für ein bestimmtes Untersuchungsziel entworfen wurde). Diese Modelle werden in speziellen Ausführungsumgebungen (üblicherweise in Simulationssystemen) ausgeführt, um Modellergebnisse zu erhalten. Systematisches Experimentieren bei Variation der Modelleingabeparameterbelegung führt in der Regel zu sehr vielen, potentiell lang andauernden Experimenten, die geplant, dokumentiert, automatisiert ausgeführt, überwacht und ausgewertet werden müssen.

Häufig besteht dabei das Problem, dass dem Experimentator (der üblicherweise kein Informatiker ist) adäquate Ausdrucksmittel fehlen, um seine Experimentier-Prozesse formal zu beschreiben, so dass sie von einem Computersystem automatisiert ausgeführt werden können. Dabei müssen Verständlichkeit, Nachnutzbarkeit und Reproduzierbarkeit (zusammengefasst als *Transparenz*) gewahrt werden.

Ein existierender Ansatz, um dieses Problem zu adressieren, besteht in der Verwendung eines *Scientific-Workflow-Management-Systems (S-WfMS)*. Das ist anwendbar, weil der Prozess des Experimentierens als eine Spezialisierung eines wissenschaftlichen Prozesses aufgefasst werden kann, der eine spezifische Struktur aufweist. Diese Struktur besteht aus den geordneten Phasen Planung, Ausführung und Auswertung von Experimenten. Eine Formalisierung eines solchen Prozesses wird in dieser Arbeit als *Experimentier-Workflow* bezeichnet. Dabei handelt es sich gleichzeitig um eine Spezialisierung eines Scientific-Workflows. Ein S-WfMS ist flexibel genug, um auch Experimentier-Workflows zu handhaben, allerdings verwenden diese Systeme dafür lediglich Konzepte wie Daten- und Kontrollfluss. Das System hat keine Kenntnis über die Semantik der Daten und wie diese in Beziehung zum zu untersuchenden Modell oder dem Ziel des Experimentierens stehen. Dies kollidiert mit der zielorientierten Denkweise der Experimentatoren.

Der in dieser Arbeit dargestellte, neue Ansatz besteht darin, generelle Experimentier-Workflow-Konzepte zu identifizieren und diese als eine metamodellbasierte *Domain-Specific-Language (DSL)* zu formalisieren, die hier als *Experimentation-Language (*ExpL*)* bezeichnet wird. ExpL beinhaltet allgemeine Workflow-Konzepte, wie Kontrollfluss und die Komposition von Aktivitäten, und erlaubt das Modellieren von Experimentier-Workflows auf einer frameworkunabhängigen, konzeptuellen Ebene. Dieser Ansatz basiert auf der Idee, das Paradigma von Model-Driven-Engineering auch auf die Domäne der Experimentier-Workflows anzuwen-

den. Dadurch werden die Nachnutzbarkeit und das Publizieren von Experimentier-Workflows nicht mehr durch die Gebundenheit an ein spezielles Framework behindert. Zudem ist es einfacher, das Computersystem zu wechseln, mit dem das Modell ausgeführt wird, weil die Experimentierbeschreibung nicht neu erstellt werden muss. Das ist sinnvoll, um andere Untersuchungsziele zu verfolgen, die nicht von diesem Computersystem unterstützt werden. Die Beschreibungsart des Modells muss dies jedoch unterstützen.

ExpL beinhaltet allgemeine Workflow-Konzepte, wie Aktivitäten und Ressourcen, sowie deklarative Elemente, um beispielsweise Parameterstudien besser unterstützen zu können. Diese Neuheit hilft dabei, die angemessene Abstraktionsebene zur Beschreibung von Experimenten dem Experimentator zur Verfügung zu stellen. ExpL wird immer in einer konkreten Experimentierdomäne benutzt (z. B. innerhalb der Geographie oder Physik), die spezifische Anforderungen an Konfigurations- und Auswertemethoden aufweist. Um mit dieser Domänenspezifik umzugehen, wird in dieser Arbeit vorgeschlagen, diese beiden Aspekte separat in zwei weiteren, abhängigen *Domain-Specific-Languages (DSLs)* zu behandeln: für *Konfiguration* und *Auswertung*. Beispielsweise besitzt eine *Konfigurations-DSL* Konzepte zur Beschreibung von Datenstrukturen, Operatoren und Einheiten. Eine *Auswertungs-DSL* sollte u. A. Ausdrucksmittel zur Formulierung von Metrik-Funktionen und für die Anwendung von Optimierungsverfahren zur Verfügung stellen.

Der Einsatz von ExpL bietet verschiedene Vorteile im Vergleich zu existierenden Ansätzen: Spezifisch zugeschnittene Ausdrucksmittel im Vokabular des Experimentators vereinfachen die Verwendung von ExpL im Vergleich zu S-WfMS. Das Vokabular von ExpL bietet Unterstützung für alle Aspekte eines Experimentier-Workflows, insbesondere für eine konsistente Dokumentation unter Einbeziehung der Herkunft aller Artefakte. Dies führt zu einem verbesserten Verständnis der Experimente, so dass diese mit weniger Aufwand für andere Wissenschaftler publiziert werden können. Dadurch sind diese in der Lage, die Ergebnisse in einem reproduzierbar, formal beschriebenen Prozess erneut erzielen zu können, auch, wenn sie keine Informatiker sind.

In dieser Arbeit wurden drei interdisziplinäre Fallstudien bezüglich der Formalisierung von Experimentier-Prozessen aus drei unterschiedlichen Domänen durchgeführt. Diese Fallstudien beschäftigen sich mit dem Experimentieren auf der Basis folgender Modellarten: einem Landnutzungswandelmodell (SLEUTH), einem Modell zur Abstraktion eines Sensornetzwerkes zur Erdbebenfrühwarnung (SOSEWIN) und einem Modell zur Beschreibung optischer Nanostrukturen.

Inhaltsverzeichnis

Teil I.

Einführung und Grundlagen

Wenn die Sprache nicht stimmt, so ist das, was gesagt wird, nicht das, was gemeint ist.

Konfuzius

1. Überblick der Arbeit

Das Thema dieser Arbeit ist eine Formalisierung des Experimentierens auf der Basis von computerbasierten Modellen. Ein *computerbasiertes Modell* ist dabei eine Abstraktion eines (realen oder imaginären) Systems, das durch dieses Modell in Verhalten und Struktur beschrieben wird und von einem Computer verarbeitet werden kann.[1] *Experimentieren* auf der Basis eines computerbasierten Modells bedeutet, dieses Modell (stellvertretend anstelle des Systems) mit einem Computersystem auszuführen, wobei das Modell mit Eingaben versorgt wird und Ausgaben produziert, die durch das modellierte Verhalten definiert sind. Ein solches Verfahren bezeichnet man als *Modellexperiment* bzw. als *Simulation*. Modelle, die simulativ untersucht werden, sind oftmals zu komplex, als dass ihre Ausgaben durch andere Verfahren (z. B. aus der Analysis) berechnet werden könnten. *Formalisierung* bedeutet in diesem Kontext, den Prozess des Experimentierens mit Hilfe der in dieser Arbeit neu entwickelten (und prototypisch implementierten), domänenspezifischen Computersprache *Experimentation-Language (*ExpL*)* zu beschreiben.

An dieser Stelle soll das Verständnis durch ein kleines Beispiel gefördert werden, das schon fast klassisch zu nennen ist: Der *Barbershop*[2] (dt. Friseursalon). Traditionell arbeitet ein Friseur (*barber*) namens Joe allein in seinem Salon, und er kann nur einen Kunden gleichzeitig bedienen. Kunden betreten in zufälligen Zeitabständen den Salon. Ein Kunde kann sofort bedient werden, falls Joe gerade mit keinem anderen Kunden beschäftigt ist. Andernfalls reiht er sich in die Warteschlange wartender Kunden ein. Ein Kunde benötigt Joes Dienste für eine bestimmte, zufällige Zeit. Danach verlässt er den Salon. Abbildung 1.1 auf der folgenden Seite visualisiert die geschilderten Zusammenhänge.

Aus dieser informalen Systembeschreibung kann unter Berücksichtigung eines Untersuchungszieles ein Modell des Systems erstellt werden. Beispielhaft soll die Frage untersucht werden, wie lange Kunden durchschnittlich warten müssen, bis sie bedient werden.[3] Dabei soll weiter angenommen werden, dass Kunden nur innerhalb einer bestimmten Zeitdauer, der Öffnungszeit des Salons, eintreffen und jeder Kunde bedient wird, der innerhalb dieser Zeit den Salon betreten hat. In den Modellexperimenten sollen verschiedene Eingabeparameterbelegungen

[1] Der Begriff *Modell* wird in dieser Arbeit nur im Sinne eines computerbasierten Modells verwendet (siehe Abschnitt 2.1.3). In dieser Einleitung ist *Modell* zudem intuitiv als *Simulationsmodell* zu verstehen – eine Herleitung der Begriffe erfolgt in Kapitel 2.

[2] Das Barbershop-Beispiel wird bereits 1974 für die Verwendung des Simulationssystems *General Purpose Simulation System (GPSS)* genutzt [Sch74]. Seitdem wird es in vielen nachfolgenden Systemen zur Demonstration verwendet.

[3] Diese einfache Fragestellung ließe sich auch analytisch beantworten. Bei komplexen Fragestellungen an komplexen Modellen ist dies i. A. nicht möglich.

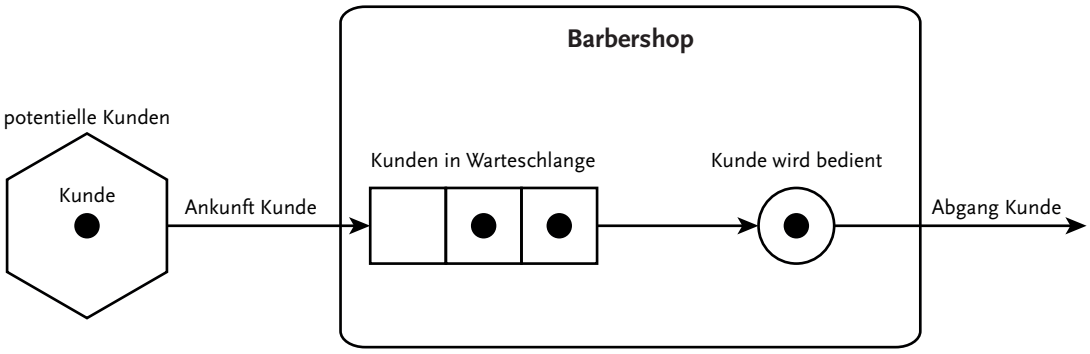


Abbildung 1.1.: Beispiel Barbershop

für das Modell variiert werden können, wie beispielsweise die Gesamtanzahl der in einer bestimmten Zeit eintreffenden Kunden und die Zeitdauer der Behandlung eines Kunden, die innerhalb eines minimalen und eines maximalen Wertes gleichverteilt sein soll.

Angenommen, Joe sieht in den Ergebnissen dieser ersten Modellexperimente, dass seine Kunden im Durchschnitt zu lange warten müssen. Er überlegt sich weiter, dass bei einer längeren Öffnungszeit die durchschnittliche Wartezeit sinken sollte (sofern die Anzahl der Kunden konstant bleibt und sie auch keine länger andauernden Behandlungen in Anspruch nehmen). Dies soll als Hypothese[4] dienen, die in weiteren Modellexperimenten zu überprüfen ist. Andere Hypothesen sind beispielsweise, ob anstelle längerer Öffnungszeiten nicht ein weiterer Mitarbeiter eingestellt werden könnte oder ob sich die Zeitdauer pro Kunden durch effizienteres Arbeiten verkürzen ließe.

Abbildung 1.2 zeigt vereinfacht den allgemeinen Ablauf des Experimentierens in Form eines *Experimentier-Workflows*. Ein Experimentier-Workflow ist eine Spezialisierung eines Scientific-Workflows und besteht aus einer zielgerichteten Abfolge von automatisierten Aktivitäten, die den Prozess des Experimentierens beschreiben.[5] Beim Experimentieren wird vom Untersuchungsziel und einer möglicherweise darauf aufbauenden Hypothese ausgegangen. Wie das Barbershop-Beispiel zeigt, kann Experimentieren zunächst auch darauf abzielen, erst das Formulieren einer Hypothese zu ermöglichen. Experimentieren ist somit allgemein ein zyklischer Prozess, bei dem Experimentergebnisse Einfluss auf die Planung weiterer Experimente nehmen.

Ein *Experiment* ist dabei durch eine konkrete Konfiguration gekennzeichnet, die sowohl die Eingabedaten des Modells, als auch Parameterwerte für die Ausführung des Modells einschließt. Es umfasst die Erstellung einer solchen Konfiguration, die Ausführung eines Modells und auch die Auswertung der Modellergebnisse (siehe Abbildung 1.2). Dabei wird vorausgesetzt, dass ein Modell zur Verfügung steht, mit dem sich das Untersuchungsziel erreichen

[4] Eine *Hypothese* ist eine Aussage, die es zu überprüfen gilt. Sie ist an bestimmte Annahmen gebunden.

[5] In Abbildung 1.2 werden manuelle Aktivitäten (z. B. „Hypothese formulieren“) einbezogen. Diese müssen, zumindest partiell, automatisiert werden, um der in Abschnitt 4.1 gegebenen, strengeren Definition eines (Experimentier-)Workflows zu entsprechen.

lässt (die Erstellung von Modellen ist nicht Gegenstand dieser Arbeit). Die Prozesse von Modellierung und Experimentieren mit dem erstellten Modell sind eng miteinander verbunden und werden als *Modellierung & Simulation (M&S)* [ZPK00] bezeichnet. Eine Verbindung ist das Untersuchungsziel, das bei beiden Prozessen eine zentrale Rolle spielt. Zudem geschieht nicht nur das Experimentieren gemeinhin iterativ, sondern auch die Modellbildung, bei der Experimentergebnisse wiederum zu verbesserten Modellen führen.

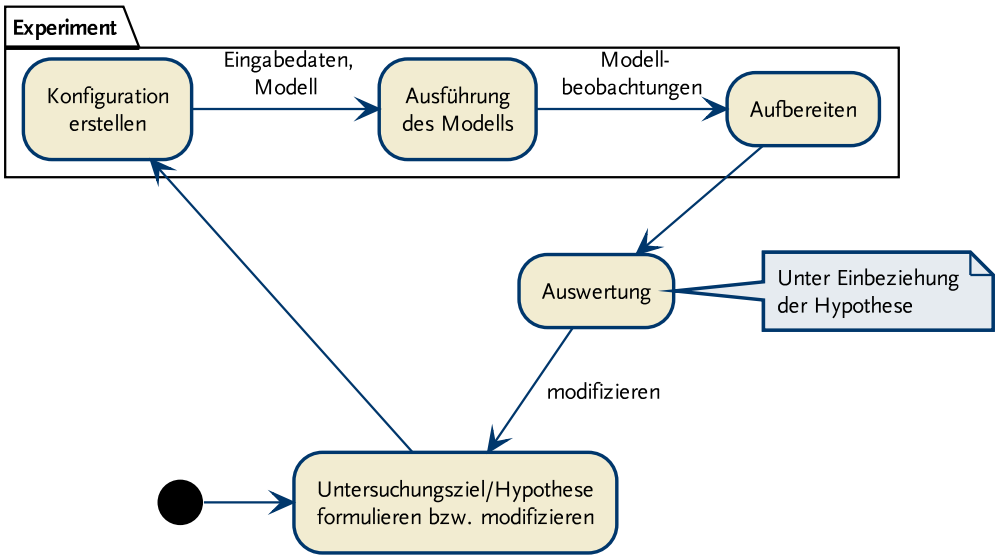


Abbildung 1.2.: Einfacher Experimentier-Workflow (als UML-Aktivitätsdiagramm)

1.1. Motivation

Modellierung & Simulation (M&S) bedeutet die Erstellung und experimentelle Untersuchung von computerbasierten Modellen, die Struktur und Verhalten eines Systems abstrahiert beschreiben. M&S ist ein anerkanntes und sehr wichtiges Instrument, um spezifische Aussagen über Systeme verschiedenster Art zu treffen. Oftmals sind dabei die betrachteten Systeme entweder noch gar nicht physisch existent oder durch anderweitige Gründe einer realen Untersuchung schwer zugänglich (z. B. könnten die zu betrachtenden Systeme in der realen Umsetzung zu komplex, zu aufwändig, zu teuer oder zu gefährlich sein). Deshalb untersucht man stellvertretend für das System ein Modell dieses Systems, wobei dieses Modell im Grad seiner Abstraktion dem Untersuchungsziel angemessen sein muss.

In vielen naturwissenschaftlichen Disziplinen ist M&S daher ein unverzichtbares Instrument des Erkenntnisgewinns. Nur ein Beispiel ist das Katastrophenmanagement, das im Informatik-Graduiertenkolleg *Modellbasierte Entwicklung von Technologien für selbstorganisierende dezentrale Informationssysteme im Katastrophenmanagement (METRIK)* mit neuen Techniken unterstützt wird. Die beteiligten Disziplinen im Katastrophenmanagement sind je nach Blickwinkel vielfältig. So bestehen in METRIK Kooperationen zu Geophysikern, Seismologen und

Geographen. Eine derartige interdisziplinäre Zusammenarbeit bietet Chancen, Expertenwissen zu kombinieren, um Basisdienste für die Frühwarnung vor Katastrophen zu entwickeln. Dabei zeigt sich, dass in den verschiedenen Disziplinen, je nach Untersuchungsziel und Abstraktionsgrad, verschiedene Modelle benötigt werden. Zudem kommen oftmals nicht nur verschiedene Modellierungstechniken, sondern auch verschiedene Modellierungs- und Simulationswerkzeuge zum Einsatz.

In einer Fallstudie dieser Arbeit wird ein Projekt zur Entwicklung eines neuartigen Systems für die Erdbebenfrühwarnung vorgestellt (siehe Abschnitt 7.3). Dabei spielen verschiedene Modelle eine Rolle, beispielsweise für die Ausbreitung von Erdbebenwellen und für das Frühwarnsystem selbst, die von verschiedenen Experten aus der Geophysik und der Informatik erstellt wurden. Nur in der Kombination dieser (Teil-)Modelle kann eine experimentelle Untersuchung stellvertretend für das Gesamtsystem vorgenommen werden. Das setzt, neben der genauen Kenntnis von Zweck und Grenzen der Teilmodelle, voraus, den Prozess des Experimentierens zu verstehen und nachvollziehen zu können, durch welche Aktionen welche Ergebnisse produziert wurden. Das kann nur durch eine Dokumentation des Experimentier-Prozesses gelingen, die auch für solche Domänen-Experten[6] verständlich ist, die keine Informatiker sind.

Die interdisziplinäre Zusammenarbeit von Forschern ist insbesondere erforderlich, wenn es um Arbeiten auf dem Weg zur Realisierung von Visionen geht. Eine solche Vision sind beispielsweise *Smart Cities* [JZH+12]. Dabei werden allgemein Technologien erforscht, die das Leben von Stadtbewohnern verbessern sollen. Diese Technologien sind sehr vielfältig und zu ihnen gehören auch Sensorsysteme, die Lärm, Luftverschmutzung und Temperaturen lokal überwachen, um Warnungen oder konkrete Maßnahmen in bestimmten Straßen- oder Häuserzügen zu ermöglichen. Solche Systeme sollen möglichst unauffällig, selbstorganisierend und fehlertolerant im Hintergrund arbeiten. Weil es zu aufwändig und zu teuer wäre, ein solches System zu Forschungszwecken in der Realität aufzubauen, werden sie typischerweise zunächst in Modellexperimenten auf der Basis von verschiedenen Modellen untersucht und durch Experimente mit realen Sensorknoten im Labor ergänzt.

Eine verständliche Beschreibung des Experimentier-Prozesses allein genügt jedoch nicht. Experimentieren als wissenschaftliche Methode erfordert es, den Experimentier-Prozess reproduzierbar zu beschreiben, um anderen Wissenschaftlern zu ermöglichen, ihn zu wiederholen und die gleichen Ergebnisse zu erhalten. Eine Beschreibung des Experimentier-Prozesses hat demnach hauptsächlich zwei übergeordnete Ziele:

Transparenz Experimentieren ist eine wissenschaftliche Methode. Deshalb muss der Experimentier-Prozess lückenlos dokumentiert und *reproduzierbar* sein, sowie derart *verständlich*, dass er mit wenig Aufwand in der wissenschaftlichen Gemeinschaft publiziert und nachvollzogen werden kann. Dies impliziert eine möglichst *intuitive* Art, Experimente formal beschreiben zu können und schließt insbesondere das Untersuchungsziel mit ein.

[6] Domänen-Experten besitzen fundiertes Wissen über die Domäne, in der sie Modelle erstellen. Oft sind sie auch in der Rolle des Experimentators und führen Experimente mit den von ihnen erstellten Modellen durch.

Automatisierbarkeit Der Prozess des Experimentierens muss durch ein Computersystem automatisierbar sein, um die oftmals notwendige große Anzahl von Experimenten beherrschbar zu machen und Fehler bei der Ausführung zu minimieren. Dabei soll sich der Experimentator auf den wissenschaftlichen Gehalt seiner Experimente konzentrieren können und sollte möglichst wenig Aufwand für Routine-Tätigkeiten in der Durchführung seiner Experimente betreiben müssen.

1.2. Problemstellung

In M&S vertraut man darauf, dass das modellierte Verhalten (zumindest bezüglich des Untersuchungszieles) dem tatsächlichen Verhalten des modellierten Systems entspricht. Dieses Vertrauen muss dann sukzessive durch Experimente gestärkt werden – generell beweisen lässt sich die Verhaltensäquivalenz von Modell und System nicht. Umso wichtiger sind das Design und das Management von Modellexperimenten. Sie verursachen in der Praxis jedoch Herausforderungen bzw. Probleme, die man grob in Daten- und Ressourcen-Probleme einteilen kann. So erfordern aussagekräftige Modellexperimente das systematische Untersuchen des Modelleingabeparameterraumes, wodurch sehr viele[7] Modellexperimente geplant, durchgeführt, überwacht und ausgewertet werden müssen. Allein das Ausführen des Modells kann viele Ressourcen, wie Rechenzeit und Speicherplatz in Anspruch nehmen. So benötigt beispielsweise bereits ein Modellexperiment eines drahtlosen Maschennetzwerkes, bei dem Aspekte des Sendens und Empfangens von Nachrichten zwischen Netzwerkknoten betrachtet werden, typischerweise mehrere Tage bis Wochen (natürlich in Abhängigkeit der konkreten Konfiguration, wie z. B. der Netzwerkgröße). Um dennoch innerhalb praktikabler Zeiträume Ergebnisse zu erreichen, und sofern es die Art der Modellexperimente (und des Modells) zulässt, führt man diese parallel auf verschiedenen Computersystemen aus. Dadurch ergibt sich jedoch ein größerer Bedarf an Hardware und ein höherer Planungsaufwand, um die vorhandenen Hardware-Ressourcen effektiv auszunutzen. Sobald zudem mehrere Experimentatoren die selben Hardware-Ressourcen nutzen möchten, ist der Planungsaufwand nicht mehr trivial handhabbar. Im Grid-Computing (und neuerdings im Cloud-Computing) ist das sogenannte *Scheduling*, also die Zuweisung und Durchführung von Aufgaben auf Hardware-Ressourcen zu bestimmten Zeiten, ein wesentlicher Teil der Forschung auf diesem Gebiet.

Die Bereitstellung geeigneter Modelleingabedaten ist ebenfalls herausfordernd. Oftmals müssen diese Daten erst aus Rohdaten aufbereitet werden, die zudem nicht selten aus verschiedenen, heterogenen Datenbeständen hervorgehen. So stammen beispielsweise die Erdbebensensordaten aus der SOSEWIN-Fallstudie, die in dieser Arbeit durchgeführt wurde, sowohl aus Textdateien unterschiedlichen Formates, als auch aus relationalen Datenbanken. Dabei war zudem die Datenqualität sehr unterschiedlich, was ein generelles Problem ist, denn die Modelleingabedaten müssen i. A. eine einheitliche Qualität aufweisen. Neben fehlender Datenqualität zeigt sich oftmals, dass bestimmtes Datenmaterial gänzlich fehlt. In einem solchen

[7] Das Kreuzprodukt aller Mengen von Eingabeparameterwerten wird gebildet, und jedes einzelne Modellexperiment verwendet genau ein Element dieser Kreuzproduktmenge als Eingabe.

Fall kann man möglicherweise fehlendes Datenmaterial aus anderen Quellen unter Zuhilfenahme von Expertenwissen ableiten, oder man muss das zugrundeliegende Modell modifizieren. Eine solche Modellmodifikation ist nicht trivial und üblicherweise ist ein Vergleich mit dem unmodifizierten Modell notwendig (wie z. B. in der SLEUTH-Fallstudie in dieser Arbeit).

Eine technische Herausforderung ist die Bereitstellung der Modelleingabedaten zur richtigen Zeit am richtigen Ort, an dem sie benötigt werden. In einem verteilten System zur Ausführung von Modellexperimenten (wie z. B. im Grid- oder Cloud-Computing) müssen die Modelleingabedaten lokal auf dem Knoten verfügbar sein, der das Modellexperiment ausführt. Je nach der Größe der Modelleingabedaten kann hierbei der lokale Speicher nicht ausreichend sein, oder die Bandbreite zum Streamen der Daten während der Experimentausführung kann zu gering sein.

Nicht nur zur Aufbereitung der Modelleingabedaten ergeben sich Ketten von Datenverarbeitungsaufgaben, sondern auch nach der Modellausführung werden die Modellbeobachtungsdaten oftmals aggregiert und statistisch sowie grafisch ausgewertet. Jede dieser Aufgaben erfordert spezifische Werkzeuge, die zudem in verschiedenen Konfigurationen und Versionen genutzt werden können. Zusätzlich bestimmt die Reihenfolge, in der die Werkzeuge ausgeführt werden, das Ergebnis der Auswertung, weshalb oftmals eine Variation in ihrer Ausführungsabfolge ein Teil des wissenschaftlichen Erkenntnisgewinnungsprozesses ist.

Aus dem bisher gewonnenen Blickwinkel erscheint das Experimentieren als ein sich dynamisch verändernder, iterativer Prozess mit variabel komponierbaren Teilaufgaben (die von spezifischen Werkzeugen ausgeführt werden). Ein solches Szenario kann auf Konzepte von Workflows abgebildet werden, bei denen Kontrollflüsse (bzw. Datenflüsse) zwischen Aktivitäten bzw. den einzelnen Teilaufgaben flexibel beschreibbar sind. Das Experimentieren als wissenschaftlicher Prozess kann durch *Scientific-Workflow-Systeme*[8] gehandhabt werden, die sich auf Ablaufsteuerung und Datenflüsse zwischen Aktivitäten konzentrieren und generell sehr flexibel in der konkreten Ausprägung ihrer Aktivitäten und deren Komponierbarkeit sind. Scientific-Workflow-Systeme werden typischerweise zur Automatisierung von Datenverarbeitungsaufgaben eingesetzt, die sich sehr gut als Komposition von Aktivitäten beschreiben lassen.

Durch den Einsatz eines Scientific-Workflow-Systems wird Flexibilität gewonnen, jedoch fehlen Konzepte, um die speziellen Eigenschaften und Möglichkeiten beim Experimentieren mit Modellen zu berücksichtigen. Für Modellexperimente werden Scientific-Workflow-Systeme deshalb gewöhnlich nicht verwendet. Solch fehlende Konzepte sind u. A.:

- die Variation von Eingabeparameterbelegungen des Modells,
- Feedback zwischen Experimenten oder auch
- eine klare Trennung der Spezifikationen für die Ein- und Ausgabeschnittstelle des zu untersuchenden Modells, für seine Ausführungsumgebung und für die Experimentausführung.

[8] Beispiele für Scientific-Workflow-Systeme sind Taverna [OAF+04], Kepler [LAB+06] und Pegasus [DSS+05] (siehe Abschnitt 5.5).

In einem traditionellen Scientific-Workflow-System wäre beispielsweise die Ausführung des zu untersuchenden Modells nur indirekt und damit intransparent ersichtlich, weil es typischerweise in Form eines Binärprogramms (des Simulators) in einer Aktivität versteckt aufgerufen werden würde. Eine Änderung dieses Binärprogramms kann durch eine Änderung am zugrundeliegenden Modell bedingt sein, muss es aber nicht (z. B. kann nur die Ausführungsumgebung des Modells geändert worden sein). Ob ein anderes Modell verwendet wurde, ist innerhalb der Workflow-Beschreibung somit nicht ersichtlich. Damit ein Experiment verständlich und seine Ergebnisse nachvollziehbar sind, müssen diese Zusammenhänge jedoch auf konzeptioneller Ebene berücksichtigt werden. Durch die Flexibilität von Scientific-Workflow-Systemen lassen sich diese Nachteile zwar grundsätzlich ausgleichen, indem neue Arten von Aktivitäten implementiert werden. Doch dadurch würde sich auch die Anzahl an Aktivitäten erhöhen, die ein Experimentator kennen und in seinem Experimentier-Workflow verwenden müsste. Entsprechend verringert sich die Übersichtlichkeit, und die Fehleranfälligkeit erhöht sich. Die Komposition von Aktivitäten ist somit nicht der richtige Abstraktionsgrad – der Experimentator benötigt Konzepte bzw. Ausdrucksmittel auf einer höheren Ebene, mit denen er solche Aktivitätskompositionen abstrahiert beschreiben kann.

Bei Simulationssystemen[9] liegt der Fokus auf der Modellierung (Scientific-Workflow-Systeme bieten keine Modellierungsunterstützung). Simulationssysteme sind deutlich weniger flexibel als Scientific-Workflow-Systeme und sind der historisch ältere Ansatz für das Experimentieren mit Modellen. Sie bieten teilweise eingeschränktes Management von Modellexperimenten hinsichtlich Planung, Ausführung und Auswertung an. Dabei handelt es sich um integrierte Lösungen, deren Einschränkungen auf konzeptioneller Ebene liegen: So sind die Spezifikationen des Modells und des Experiment-Managements üblicherweise nicht voneinander getrennt und erfolgen spezifisch mit den Sprachmitteln und Konzepten des jeweiligen Simulationssystems.

Die Gebundenheit an spezifische Sprachmittel und Formate haben Simulationssysteme gemeinsam mit den Scientific-Workflow-Systemen – dieser geschlossene Ansatz ist eine plattformabhängige Modellierung. Heutzutage bemüht man sich jedoch, Systeme plattformunabhängig zu modellieren, indem man Paradigmen wie z. B. dem des *Model-Driven-Engineering (MDE)*[10] folgt. Die plattformunabhängige Modellierung hat Vorteile bezüglich Wiederverwendung und Transparenz von Modellen. Dies ist beispielsweise dann besonders vorteilhaft und erwünscht, wenn ein plattformunabhängiges Modell eines Systems mittels verschiedener Simulationssysteme oder realer Testbeds untersucht werden soll. Ein solches Vorgehen hilft, jeweils spezifische Fragestellungen beantworten zu können, die nur in einer bestimmten Plattform (Ausführungsumgebung für das Modell) geklärt werden können. Simulationssysteme bieten keine Unterstützung für MDE, wohingegen die Flexibilität von Scientific-Workflow-Systemen dazu genutzt werden kann, einen MDE-Ansatz umzusetzen. Entsprechend sind Simulationssysteme vergleichsweise wenig flexibel im Hinblick auf den Experimentier-

[9] Beispiele für Simulationssysteme sind SLX, JiST/SWANS, DESMO-J und ODEMx, auf die in Kapitel 3 eingegangen wird.

[10] MDE ist ein weitgefasster Begriff für die modellbasierte Entwicklung, der insbesondere die Komplexität von Plattformen und domänenspezifischen Konzepten berücksichtigt. Eine präzisere Begriffsklärung und die Relationen zwischen den Begriffen MDE, MDD und MDA wird in Abschnitt 9.1.1 auf Seite 119 gegeben.

Prozess: Die komponierbare Ausführung von Teilaufgaben im Sinne eines Workflows unter Integration externer Werkzeuge wird nicht unterstützt. Das ist hinsichtlich der häufig für Experimente notwendigen Datenvor- und Datennachverarbeitung problematisch.

In einem Satz zusammengefasst lässt sich die in dieser Arbeit betrachtete *Problemstellung* wie folgt formulieren:

> Dem Experimentator fehlen geeignete Beschreibungsmittel und Computersysteme, um seine Experimentier-Prozesse auf der Basis von computerbasierten Modellen transparent und automatisierbar zu formalisieren und auszuführen.

1.3. Ziel

Für die Formalisierung des Experimentierens mit computerbasierten Modellen wird in dieser Arbeit ein neuer Ansatz entwickelt. Die dafür notwendigen Konzepte sind auf der Basis von Analysen des M&S-Prozesses und Fallstudien erarbeitet worden. Ein Teil dieser Konzepte findet sich im existierenden Ansatz der Scientific-Workflows als Formalisierung eines wissenschaftlichen Prozesses wieder (z. B. Kontrollflüsse zwischen Aktivitäten und deren Komponierbarkeit). Weitere Konzepte sind spezifisch für das Experimentieren mit computerbasierten Modellen (z. B. die Variation von Modelleingabeparameterbelegungen). Eine Kombination dieser Konzepte führt zu einer neuen, speziellen Klasse von Scientific-Workflows, den *Experimentier-Workflows*. Durch den Einsatz von Experimentier-Workflows lässt sich die geforderte Automatisierbarkeit des Experimentier-Prozesses erreichen. Die zusätzlich geforderte Transparenz wird jedoch vor allem durch eine geeignete Beschreibungsart der Experimentier-Workflows ermöglicht. Hierfür wird die in dieser Arbeit neu entwickelte und prototypisch implementierte, domänenspezifische Sprache ExpL angewendet. ExpL vereint die identifizierten Konzepte, und ermöglicht es, die Hypothese dieser Arbeit zu überprüfen.

Die dieser Arbeit zugrunde liegende *Hypothese* lautet:

> Das Experimentieren mit computerbasierten Modellen kann transparent und automatisierbar beschrieben werden, indem man geeignete (und erweiterte) Workflow-Konzepte mit Hilfe einer domänenspezifischen Sprache formalisiert und diese Sprache mittels eines Computersystems anwendet.

Folglich lautet das Ziel dieser Arbeit, diese Hypothese zu überprüfen. Hierfür wurden die domänenspezifische Sprache ExpL und zugehörige Sprachwerkzeuge[11] entwickelt und in Fallstudien und Beispielen angewendet.

1.4. Verwandte Arbeiten

Der neue Ansatz in dieser Arbeit liegt in der Kombination aus drei Themengebieten: *Experimentieren* mit computerbasierten Modellen, Scientific-*Workflows* und metamodellbasierten,

[11] Unter *Sprachwerkzeugen* sollen alle Werkzeuge verstanden werden, die zum Erstellen, Verwenden und Ausführen der zugehörigen Computersprache benutzt werden (z. B. intelligente Editoren, Interpreter, Compiler).

domänenspezifischen Sprachen. Diese Themengebiete sind auch die Grundlage der Struktur dieser Arbeit in Teil I (siehe auch Abschnitt 1.6 und dazu Abbildung 1.3).

Experimentieren

Die zwei Klassen von Werkzeugen (Simulationssysteme und Scientific-Workflow-Systeme), die das Experimentieren mit computerbasierten Modellen grundsätzlich unterstützen, wurden bereits in der Problemstellung genannt. Dort wurden auch die Vor- und Nachteile angesprochen, die sowohl Simulationssysteme als auch Scientific-Workflow-Systeme in dieser Hinsicht aufweisen. Zudem existieren vielfältige, eigenentwickelte Softwarelösungen für das Management von Modellexperimenten. Diese Lösungen sind an bestimmte Werkzeuge gebunden, wodurch mit ihnen beschriebene Experimente i. A. nicht auf eine andere Werkzeugsammlung übertragbar sind. Eine solche Werkzeugsammlung kann ein Simulationssystem sein (z. B. DISMO als Erweiterung des Simulationssystems DESMO-J, siehe Abschnitt 3.2.2) oder auch eine Werkzeugsammlung für ein Testbed. In die letztgenannte Kategorie fällt beispielsweise das *DES-Testbed Management System (DES-TBMS)* zum Management von Experimenten für das drahtlose Maschennetzwerk der Forschungsgruppe *Distributed Embedded System (DES)* an der *Freien Universität Berlin* [GJBW09].

Mit der grundlegenden Frage, welche Erkenntnisse durch die Forschung an sich und insbesondere welche Erkenntnisse mittels der Methodik des Experimentierens zu erlangen sind, hat sich beispielsweise Popper in *Logik der Forschung* beschäftigt [Pop05]. Um den Gegenstand des Experimentierens in dieser Arbeit – das computerbasierte Modell – einzuordnen, wird die Begriffsdefinition von Stachowiak [Sta73] herangezogen. Das Experimentieren mit Modellen stützt sich auf die Arbeit von Zeigler [ZPK00].

Workflows

In der Problemstellung wurde darauf hingewiesen, dass Scientific-Workflow-Systeme für das Experimentieren mit computerbasierten Modellen nicht typischerweise verwendet werden. Demgegenüber gibt es Arbeiten zur Verwendung von Business-Workflows für das Management von Modellexperimenten [GSK+11]. Dabei werden vor allem etablierte Standards und existierende Workflow-Management-Systeme aus der Geschäftsprozessmodellierung wiederverwendet und um Eigenschaften von Scientific-Workflows erweitert (z. B. um den Fokus auf Daten), so dass sie sich für das Experimentieren mit Modellen technologisch eignen. Durch eine solche Erweiterung von Workflow-Management-Systemen für Business-Workflows nimmt deren Komplexität weiter zu und dies könnte zu einer hohen Einstiegshürde für Experimentatoren führen. Der zentrale Unterschied zu der hier vorliegenden Arbeit liegt im Abstraktionslevel: Experimentier-Workflows und ihre Beschreibung in ExpL bieten speziell zugeschnittene Sprachmittel, so dass der Experimentator seine Experimente derart prägnant (z. B. in Teilen deklarativ) ausdrücken kann, dass sie die geforderten Eigenschaften von Transparenz und Automatisierbarkeit erfüllen. Dadurch lässt sich ExpL bereits frühzeitig im M&S-Zyklus einsetzen, wodurch sich insgesamt die Qualität der Ergebnisse potentiell verbessert.

Die grundlegende Terminologie von Workflows, die in dieser Arbeit verwendet wird, basiert auf der *Workflow Management Coalition (WfMC)* [WfM99]. Spezifische Eigenschaften von Scientific-Workflows werden beispielsweise von Deelman und Ludäscher [LAB+09, DGST09, MBZL09] genannt (siehe auch Abschnitt 4.2).

Domänenspezifische Sprachen

Domänenspezifische (Computer-)Sprachen sind eine relativ alte und viel genutzte Technologie, um computerverständliche Beschreibungen zu erstellen, die Konzepte einer konkreten Domäne verwenden.[12] Als Standardwerk auf diesem Gebiet kann *Domain-specific Languages* von Fowler betrachtet werden [FP10]. Sprachen mit Hilfe eines Metamodells zu beschreiben, wie es in dieser Arbeit getan wird, ist eine relativ neue Entwicklung in der Informatik (siehe Kapitel 6). Eine umfassende Darstellung dieser Technik bietet Scheidgen in seiner Dissertation [Sch09].

Aufgrund des weit gefassten Begriffes von domänenspezifischen Sprachen kann man etablierte Workflow-Beschreibungssprachen, wie beispielsweise XPDL[13] und BPEL, als domänenspezifische Sprachen verstehen. Demgegenüber ist die Entwicklung einer metamodellbasierten, domänenspezifischen Sprache zur Beschreibung von Experimentier-Workflows ein neuer Ansatz, der in dieser Arbeit verfolgt wird.

Relationen zu e-Science und Scientific-Data-Management

Das Ziel des Experimentierens mit Modellen ist allgemein die Erlangung von wissenschaftlichen Erkenntnissen über das modellierte System. Dabei sind nicht nur die erreichten Ergebnisse, sondern insbesondere auch die Experimentier-Prozesse, die diese Ergebnisse hervorgebracht haben, als Forschungsdaten zu betrachten. Eine einheitliche Definition von Forschungsdaten existiert nicht, sondern je nach Domäne existieren Kriterien, die Forschungsdaten nach Inhalt, Umfang, Art und Format identifizieren. Wieviel Bedeutung diesen Forschungsdaten beigemessen wird, hängt ebenfalls von der entsprechenden Domäne ab. Wissenschaftsfördernde Verbände, wie beispielsweise der deutsche Wissenschaftsrat, weisen zunehmend auf die Bedeutung von Forschungsdaten und deren Management hin, wobei auf die Entwicklung und Verbesserung von Forschungsinfrastrukturen[14] gedrungen wird [Wis12, Wis11].

[12] Im Englischen ist der Begriff *Domain-Specific-Language (DSL)* für eine domänenspezifische Computersprache gebräuchlich. Der Begriff *Sprache* wird dort, wie auch in dieser Arbeit, nur im Sinne einer *Computersprache* verwendet. Als domänenspezifische Sprachen gelten beispielsweise auch Skriptsprachen, wie die des *stream editor (sed)*, der bereits 1973/1974 entwickelt wurde.

[13] *XML Process Definition Language (XPDL)* ist eine XML-basierte Sprache zur Beschreibung von Geschäftsprozessen. XPDL wurde von der Workflow Management Coalition entwickelt.

[14] „Als Forschungsinfrastrukturen werden diejenigen teilweise einzigartigen Einrichtungen, Ressourcen und Dienstleistungen in öffentlicher oder privater Trägerschaft […] verstanden, die speziell für wissenschaftliche Zwecke errichtet, mittelfristig bis tendenziell permanent bereitgestellt werden und für deren sachgerechte Errichtung, Betrieb und Nutzung spezifische fachwissenschaftliche oder interdisziplinäre (Methoden-)Kom-

E-Science ist dabei ein Schlagwort, das für *electronic* oder auch *enhanced science* steht und quasi einen neuen Geist im Umgang mit Forschungsdaten darstellt: Wissenschaftliche Publikationen sollen stets auf die Forschungsdaten verweisen können, auf denen sie basieren. Das erfordert jedoch zuerst eine Kultur entsprechend des *Open Access*, bei der ein freier Zugang zu wissenschaftlicher Literatur und anderen Materialien durch das Internet eingefordert wird. Entsprechende Infrastrukturen müssen entwickelt werden, um solche Ansätze zu unterstützen.

Computersysteme für das Scientific-Data-Management innerhalb solcher Infrastrukturen müssen sich nach [Moo01] zumindest mit den folgenden Aspekten beschäftigen:

- Erzeugung von logischen, semantisch zusammenhängenden Einheiten von Daten, die von der physischen Speicherung abstrahieren,
- Interoperabilität, so dass die Forschungsdaten von verschiedenen Computersystemen zugreifbar sind,
- Zugriffsrechte und Urheberschaft der Daten,
- die Erhebung und Verwaltung von Metadaten,
- Persistenz der Daten, wobei insbesondere ein Zugriff über die Lebenszeit des verwaltenden Systems hinaus möglich sein muss und
- die Ableitung von Wissen und Information aus dem Datenbestand.

Das ExpL-Workflow-System, das in dieser Arbeit entwickelt wurde, kann bei verschiedenen dieser Aspekte Unterstützung anbieten. Es schafft beispielsweise durch die transparent beschriebenen Experimentier-Workflows eine logische Einheit auf der Ebene des Experimentier-Prozesses. In einem Experimentier-Prozess wird dabei üblicherweise ein bestimmtes Untersuchungsziel verfolgt. Durch die Art der technologischen Umsetzung des ExpL-Workflow-Systems können leicht andere Formate und Sichten auf den Datenbestand (innerhalb der Experimentier-Workflows) erstellt werden, so dass sich darüber auch andere Systeme mit Daten versorgen lassen.

1.5. Beiträge

Ein grundlegender Beitrag der vorliegenden Arbeit ist die *Identifikation von Anforderungen* an das Experimentieren mit computerbasierten Modellen und an ein Computersystem, das dieses Experimentieren unterstützt. Resultierend aus den erkannten Anforderungen wurden geeignete *Konzepte identifiziert* und umgesetzt, mit denen diese Anforderungen erfüllt werden können.

petenzen erforderlich sind. Ihre Funktion ist es, Forschung, Lehre und Nachwuchsförderung zu ermöglichen oder zu erleichtern.“ [Wis12, S. 15 f.]

Die Experimentier-Workflow-Beschreibungssprache ExpL

Die identifizierten Konzepte führten zur Entwicklung der metamodellbasierten, domänenspezifischen Sprache ExpL zur Beschreibung von Experimentier-Workflows. Diese Sprache und die zugehörigen Sprachwerkzeuge wurden prototypisch implementiert und in Fallstudien angewendet, um ihre Eignung zu evaluieren. Durch den sprachbasierten Ansatz ergeben sich eine ganze Reihe von Vorteilen:

- Auf die Bedürfnisse von Experimentatoren zugeschnittene Sprachmittel erleichtern das Formulieren ihrer Modellexperimente.
- Ein besseres Verständnis der Abläufe und die Konsistenz zwischen Konfiguration und Ergebnissen führt zu mehr Transparenz und Wiederverwendung von Experimentier-Workflows und ermöglicht den Austausch mit anderen Wissenschaftlern sowie die Konzentration auf wissenschaftliche Fragestellungen und nicht auf technologische.
- Das vereinfachte und verbesserte Management von Modellexperimenten führt potentiell zu einem früheren Einsatz dieser Methodik bei der Modellierung (bzw. in schnelleren und agileren Modellierungs- und Simulationszyklen), wodurch frühzeitig Probleme des Modells erkannt und behoben werden können.
- Die Möglichkeit der Kopplung von ExpL mit anderen metamodellbasierten, domänenspezifischen Sprachen, die bestimmte Aspekte einer Experimentier-Domäne erfassen. Dadurch lassen sich Experimente in dieser Domäne transparenter beschreiben, wie die Erfahrungen aus den durchgeführten Fallstudien belegen.

Mit der Bereitstellung der Sprache ExpL und dem zugehörigen ExpL-Workflow-System wird in dieser Arbeit u. a. die Transferleistung erbracht, das MDE-Paradigma auf die Spezifikation von Experimentier-Workflows anzuwenden und diese somit plattformunabhängig zu beschreiben. Dadurch kann man Experimente auf gleicher Abstraktionsebene beschreiben, auf der auch ein potentiell MDE-konformes Modell des Systems vorliegt. Dies schließt die Lücke zwischen existierenden Lösungen zur plattformunabhängigen Modellierung und bisher fehlenden, plattformunabhängigen Beschreibungsmöglichkeiten für Experimentier-Workflows.

Fallstudien

Die angesprochenen drei Fallstudien stammen aus den verschiedenen Domänen Seismologie, Geographie und Physik. Sie waren nur in Kooperation mit den jeweiligen Domänenexperten möglich. In allen drei Fallstudien ist ein MDE-Ansatz umgesetzt worden. Die Kurzbezeichnung jeder Fallstudie leitet sich anhand des darin untersuchten Modells ab. Im Überblick sind die Fallstudien (eine detaillierte Einführung liefert Kapitel 7):

SLEUTH-Fallstudie In dieser Fallstudie mit der Geographie wurde der Landnutzungswandel für eine spezifische Region modellbasiert untersucht, wobei eine etablierte Modellklasse verwendet wurde (SLEUTH[15]). Hierfür notwendige Daten über diese Region waren jedoch nicht verfügbar, weshalb eine Re-Implementierung und Erweiterung von

[15] SLEUTH steht für *Slope, Land-cover, Exclusion, Urbanization, Transportation, Hillshade* und bezeichnet eine

SLEUTH vorgenommen wurde [LTK+12]. Diese Re-Implementierung und Erweiterung wurde mit Hilfe der domänenspezifischen Sprache *ECAL* [TKK+10, TDF09] zur Beschreibung zellulärer Automaten erreicht. Diese Fallstudie zeigt insbesondere eine Sprachkopplung von ECAL, ExpL und einer einfachen Sprache für georäumliche Auswerteverfahren (GIS-DSL) [KTF09].

Nano-Fallstudie In der Grundlagenphysik werden derzeit die Eigenschaften von optischen Nanostrukturen erforscht. Ein Beitrag im Rahmen dieser kooperativen Fallstudie zwischen der Physik und der Informatik besteht in der Erstellung und Anwendung einer domänenspezifischen Sprache zur Beschreibung von optischen Nanostrukturen (Nano-DSL) und den Experimenten mit diesen Strukturmodellen (siehe [WSKF11] und Abschnitt 7.2).

SOSEWIN-Fallstudie SOSEWIN[16] ist ein neuartiges, drahtloses, vermaschtes, dezentrales Netzwerk zur Erdbebenfrühwarnung in der Seismologie [FPM+09]. Diese Fallstudie ist die umfangreichste der hier dargestellten drei Fallstudien, weil SOSEWIN im Rahmen von zwei internationalen Forschungsprojekten entwickelt wurde. Es existieren verschiedene, plattformspezifische Modelle: Für drei Kategorien von Simulationen (Einzelknoten, Kooperation von Knoten und auf Netzwerkebene) und zur Installation auf dem realen Netzwerk. Diese plattformspezifischen Modelle basieren auf dem selben, plattformunabhängigen Modell (MDE-Ansatz).

1.6. Struktur

Nachdem in diesem Kapitel 1 ein Überblick der Arbeit gegeben wurde, beschäftigen sich die folgenden Kapitel zwei bis sechs des Teil I mit den Grundlagen[17], die zum weiteren Verständnis der Arbeit notwendig sind. Dabei werden die drei großen, inhaltlichen Themengebiete dargestellt, in deren Schnittmenge das Thema dieser Arbeit angesiedelt ist (siehe Abbildung 1.3 auf der folgenden Seite):

1. *Experimentieren* (Kapitel 2 und 3)
   In Kapitel 2 wird erläutert, was Experimentieren mit computerbasierten Modellen bedeutet. Dazu werden die Begriffe *System* und *Modell*, sowie *Experiment* und *Simulation* eingeführt. Es folgt eine Charakterisierung des Experimentierens als Teilprozess von Modellierung & Simulation. Kapitel 3 erläutert Simulationssysteme, die traditionellerweise Unterstützung für den Prozess von Modellierung & Simulation bieten.

---

etablierte Modellklasse zur Beschreibung urbanen Landnutzungswandels (siehe [Cla08, CHG97] und Abschnitt 7.1 auf Seite 71).

[16] SOSEWIN bedeutet *Self-Organizing Seismic Early Warning Information Network* und wurde im Rahmen des EU-Forschungsprojektes *Seismic eArly warning For EuRope (SAFER)* innerhalb von drei Jahren erstellt und im BMBF-geförderten Projekt *Earthquake Disaster Information System for the Marmara Region, Turkey (EDIM)* in der Marmara-Region (Türkei) testweise installiert (siehe Abschnitt 7.3 auf Seite 80).

[17] Es wird vorausgesetzt, dass der Leser grundlegendes Wissen über Grammatiken und UML-Klassendiagramme besitzt. Für eine Einleitung zu diesen Themen wird auf [HMU79] und [Lar04] verwiesen.

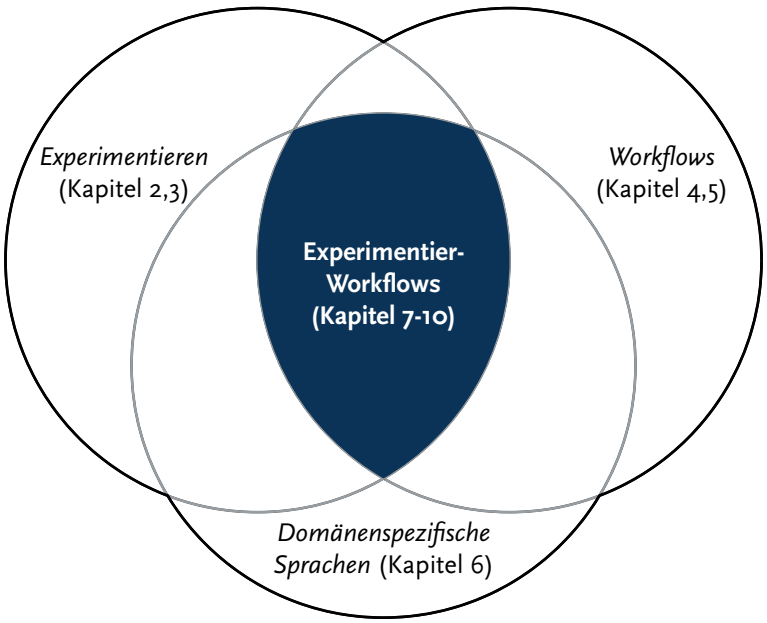


Abbildung 1.3.: Struktur der Arbeit: Experimentier-Workflows in der Schnittmenge dreier Themengebiete

2. *Workflows* (Kapitel 4 und 5)
   Das zweite, große Themengebiet dieser Arbeit sind Workflows, wobei zuerst eine Begriffsklärung in *Definition und Arten von Workflows* erfolgt und danach auf spezifische Eigenschaften und Grundbegriffe von *Workflow-Systemen* eingegangen wird. Insbesondere wird in Abschnitt 4.3 der neuentwickelte Begriff *Experimentier-Workflows* definiert.
3. *Domänenspezifische Sprachen* (Kapitel 6)
   Die Einführung zu domänenspezifischen (Computer-)Sprachen beinhaltet zuerst eine Motivation, warum solche Sprachen allgemein und warum sie speziell in dieser Arbeit von Interesse sind. Die *Aspekte einer Sprache* beschreiben, aus welchen logischen Einheiten eine Sprache aufgebaut ist. In *Metamodellbasierte Sprachentwicklung* ist der in dieser Arbeit verwendete, spezielle Ansatz erläutert, mit dem domänenspezifische Sprachen entwickelt werden können.

In Teil I gibt es zwei Abschnitte, die Relationen zwischen den Terminologien von jeweils zwei Themengebieten darstellen. Abschnitt 5.4 stellt die Verbindung von *Modellierung & Simulation* mit *Workflows* her. Abschnitt 6.4 beschreibt Relationen zwischen den Begriffen der Themengebiete *Workflows* und *domänenspezifische Sprachen*.

Teil II umfasst vier Beitragskapitel:

1. In Kapitel 7 werden drei Fallstudien vorgestellt. Mit dem Wissen aus diesen Fallstudien wurden Anforderungen für die automatisierbare und transparente Beschreibung von Experimentier-Prozessen erarbeitet (Abschnitt 7.4). Darauf aufbauend, werden Anforderungen an ein Computersystem für Experimentier-Prozesse definiert (Abschnitt 7.5),

2. Kapitel 8 beschreibt die Konzeption der Experimentier-Workflow-Beschreibungssprache ExpL, wobei insbesondere auf die getroffenen Entscheidungen bezüglich der identifizierten Konzepte eingegangen wird,
3. Kapitel 9 stellt die Architektur des ExpL-Workflow-Systems dar und
4. in Kapitel 10 wird gezeigt, wie ExpL und das ExpL-Workflow-System in den Fallstudien angewendet wurden.

Teil III beinhaltet drei abschließende Kapitel:

1. Kapitel 11 widmet sich einer kritischen Diskussion der erreichten Resultate,
2. Kapitel 12 fasst die Ergebnisse der Arbeit zusammen,
3. worauf Kapitel 13 aufbaut und einen Ausblick auf anschließende Fragestellungen und mögliche, weiterführende Arbeiten gibt.

2. Experimentieren

2.1. Grundbegriffe

2.1.1. Experiment

Ein wissenschaftliches *Experiment*[1] ist allgemein die methodisch angelegte, zielgerichtete Untersuchung eines Phänomens zur empirischen Gewinnung von Daten über dieses Phänomen. Ein Experiment besteht aus einer Menge von *Versuchen*, welche jeweils eine einmalige Durchführung eines Experimentes darstellen.

Ein Versuch und die dabei gewonnenen, empirischen Daten werden in dieser Arbeit als *Experimentlauf* bezeichnet. Ein *Experimentplan* ist eine Vorschrift zur Beschreibung von Experimenten, so dass die einzelnen Experimente aus dem Experimentplan abgeleitet werden können. Anders formuliert, beschreibt der Experimentplan die Methodik der Untersuchung (z. B. eine systematische Variation von Parameterbelegungen).

In Verbindung mit dem Begriff des Experimentplanes wird in dieser Arbeit *Experiment* präziser gefasst und um die Eigenschaft erweitert, dass die Werte der Eingangsgrößen in einem Experiment festgelegt sein sollen. Die Menge von Experimenten, welche durch einen Experimentplan beschrieben ist, wird in dieser Arbeit als *Experimentserie* bezeichnet. Abbildung 2.1 verdeutlicht die Zusammenhänge.

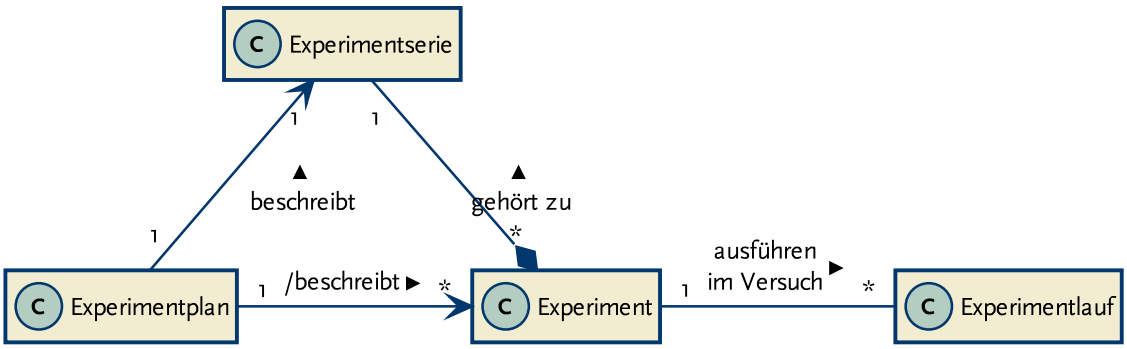


Abbildung 2.1.: Relationen der Begriffe Experimentplan, Experimentserie, Experiment und Experimentlauf (als UML-Klassendiagramm)

[1] Das Wort *Experiment* entstammt dem lateinischen *experimentum* für Versuch, Beweis, Prüfung oder Probe [Klu12].

2.1.2. System

Das Phänomen als Gegenstand einer experimentellen Untersuchung muss für das weitere Verständnis präzisiert werden: Ein Phänomen, welches bestimmte Eigenschaften aufweist, wird als *System* bezeichnet. Diese Eigenschaften werden als *Systemeigenschaften* benannt und unterscheiden sich nach Grundauffassung und Detailgrad in der Literatur. Die Auswahl der folgenden Systemdefinitionen basiert auf der Argumentation von Kunert [Kun11].

Nach Gaines [Gai79] wird ein System definiert durch die vollständige Menge seiner Bestandteile, den sogenannten *Entitäten* (oder *Systemelementen*), deren Relationen untereinander und ihre Beziehung zur Außenwelt. Zudem ist jeder wohldefinierte Teil eines Systems wiederum ein System. Ashby [Ash57] unterstreicht die Bedeutung der am System durchzuführenden, experimentellen Untersuchung (und dessen Ziel) und fordert deshalb zusätzlich eine Reduktion auf die ausschließlich dafür notwendigen Entitäten und Abhängigkeiten. Cellier [Cel91] führt diese Gedanken fort und fordert zusätzlich die Kontrollierbarkeit und Beobachtbarkeit eines Systems. Ein System muss dementsprechend Möglichkeiten zur Akzeptanz von Eingaben besitzen, auf die es mit konkreten Ausgaben reagiert. Dabei müssen sowohl Ein- als auch Ausgaben des Systems messbar sein.

Cellier hat einen zu Karplus äquivalenten Systembegriff, der in dieser Arbeit verwendet wird. Karplus fasst in [Kar77] die wesentlichen Anforderungen bei der Systembildung zusammen, aus denen sich die Systemeigenschaften ergeben:

Separabilität Ein System stellt (hinsichtlich des Systemzwecks) eine eigenständige Einheit dar und besitzt somit eine wohl definierte *Systemgrenze*. Die Systemgrenze separiert das System von der *Systemumgebung*, zu der jedes Element gehört, das keine Entität des Systems darstellt. Dabei sind Wirkungen des Systems auf die Systemumgebung und umgekehrt erlaubt. Eine Rückkopplung, bei der eine Wirkung des Systems auf die Umgebung wiederum eine Wirkung auf das System hat, ist ausgeschlossen (andernfalls wäre die Systemgrenze falsch gewählt).

Selektivität Aus allen möglichen Interaktionen und Abhängigkeiten zwischen den Entitäten des Systems und den Elementen der Umwelt (festgelegt durch die Systemgrenze) werden bei der Systembildung nur diejenigen berücksichtigt, die relevant für die durchzuführende Untersuchung sind.

Kausalität Die Eingaben des Systems sowie ein innerer Systemzustand bestimmen kausal und ausschließlich die Ausgaben. Die Ausgaben wiederum dürfen unter keinen Umständen die Eingaben beeinflussen (andernfalls sind die Systemgrenzen zu eng gewählt worden).

Das wohl wichtigste Ziel im Prozess der Systembildung ist es, die Funktion bzw. den Zweck des Phänomens durch dessen Beobachtung zu erkennen und als *Systemzweck* zu bestimmen [FA96, S. 16 f.].

Der Begriff des Experimentes kann mit Hilfe des gebildeten Systembegriffs präziser gefasst werden: Ein *Experiment* ist die zielgerichtete Untersuchung eines *Systems* zur empirischen Gewinnung von Daten über dieses *System*. Das ist analog zu Cellier, der ein Experiment als die Beobachtung der Ausgaben eines Systems versteht, während dieses mit Eingaben versorgt wird:

> An experiment is the process of extracting data from a system by exerting it through its inputs.

Experimente können grundsätzlich auf Basis des Systems selbst vorgenommen werden, oder an einem Modell des Systems (letzteres ist Gegenstand dieser Arbeit).

2.1.3. Modell

Oftmals ist es nicht praktikabel, ein Phänomen (und das daraus abstrahierte System) direkt experimentell zu untersuchen. Die Gründe hierfür können vielfältiger Natur sein und auch kombiniert auftreten:

- das System ist fiktiv, existiert also (noch) nicht (z. B. Flughafen in der Planungsphase),
- das System ist zu komplex, so dass es nicht formal beschrieben werden kann oder die formale Beschreibung lässt sich aufgrund ihrer Komplexität nicht formal analysieren (z. B. Vorhersage von Wetterphänomenen),
- das System reagiert naturgemäß zu langsam auf die Eingaben innerhalb eines Experimentes (z. B. astronomische Sonden),
- ein Experiment am System ist zu aufwendig, kostenintensiv oder (potentiell) gefährlich, um überhaupt durchgeführt oder signifikant oft wiederholt werden zu können (z. B. Untersuchungen zu Auswirkungen von Erdbeben).

Trifft mindestens einer der genannten Gründe zu, kann man sich der Technik der *Modellierung* bedienen, um eine weitere Abstraktion des Systems unter Nutzung mathematischer Formalismen vorzunehmen, das sogenannte *(System-)Modell*[2]. Allein durch den Abstraktionsprozess lassen sich bereits erste Erkenntnisse über das zu untersuchende System (und das zugrundeliegende Phänomen) erlangen.

Minsky verknüpft den Modellbegriff mit einem *Untersuchungsziel* und formuliert in [Min65]:

> To an observer B, an object A^* is a model of an object A to the extent that B can use A^* to answer questions that interest him about A.

Man kann den *observer B* mit der Rolle *Experimentator* gleichsetzen und das *object A* als *System A* verstehen. Object A^* ist aber offenbar nur ein Modell des Systems A, wenn es für das Untersuchungsziel dienlich ist. Verschiedene Untersuchungsziele können zu verschieden dafür geeigneten Modellen des selben Systems führen. Minskys Modellbegriff ist rekursiv: Ein Modell kann wiederum als System aufgefasst werden, für das eine weitere Modellbildung möglich ist.

Nach Stachowiak ist ein Modell durch mindestens drei Merkmale gekennzeichnet [Sta73, S. 131 ff.]:

Abbildung „Ein Modell ist stets ein Modell von etwas, nämlich Abbildung, Repräsentation eines natürlichen oder eines künstlichen Originals, das selbst wieder Modell sein kann."

[2] Der Begriff *Modell* bezieht sich in dieser Arbeit stets auf Systeme, daher wird keine sprachliche Unterscheidung vorgenommen.

Verkürzung „Modelle erfassen im Allgemeinen nicht alle Attribute des durch sie repräsentierten Originals, sondern nur solche, die den jeweiligen Modellschaffern und/oder Modellnutzern relevant scheinen."

Pragmatismus Modelle sind ihren Originalen nicht an sich eindeutig zugeordnet: Es ist nicht nur „die Frage zu berücksichtigen, wovon etwas Modell ist, sondern auch, für wen, wann und wozu bezüglich seiner je spezifischen Funktionen es Modell ist."

Typischerweise beschreiben Modelle die Struktur, das Verhalten und die Umgebung von Systemen.

Mathematisches Modell *Mathematische Modelle* sind in mathematischen Formeln beschriebene formale Modelle. Durch die formale Beschreibung in mathematischen Formeln kann das im Modell beschriebene Verhalten berechnet werden (d. h. welche Ausgaben es bei definierten Eingaben erzeugt). Berechenbarkeit bedeutet hierbei sowohl die Möglichkeit der analytischen Untersuchung als auch die der Approximation mittels numerischer Verfahren.

Computerbasiertes, mathematisches Modell *Computerbasierte, mathematische Modelle* (*computational models*) sind mathematische Modelle, welche aufgrund ihrer Komplexität bzw. der Anzahl ihrer Parameter typischerweise nur mit Hilfe eines Computers berechnet und ausgewertet werden können.[3] Dabei liegt das computerbasierte Modell in Form eines Programmes bzw. Programmteils vor (z. B. als Quelltext oder als Binärdatei). Dieses Programm verfügt über eine Menge von *Zustandsvariablen*[4] und *Verhaltensmethoden*. Die Wertebelegungen der Zustandsvariablen zur Laufzeit des Programms repräsentieren den *Systemzustand*. Die Verhaltensmethoden bilden das Verhalten des Systems nach, das durch die Zeit bzw. durch Ereignisse aus der Systemumgebung beeinflusst wird. In dieser Arbeit impliziert der Begriff *Modell* die Attribute *computerbasiert* und *mathematisch* (wenn nicht explizit ein anderer Kontext angegeben wird).

2.1.4. Simulation

Korn und Wait [KW78] liefern eine prägnante Definition für eine *Simulation*, die auf den bisher gegebenen Definitionen für Experiment und Modell aufbaut:

> A simulation is an experiment performed on a model.

Simulation und *Modellexperiment* (bzw. *Experiment mit einem Modell*) sind nach dieser Definition synonyme Begriffe und beziehen sich auf ein Modell, das als *Simulationsmodell* bezeichnet wird. Eine *Computersimulation* ist der Spezialfall einer Simulation, bei welcher das

[3] Für ein computerbasiertes, mathematisches Modell wird in der Literatur gelegentlich auch der Begriff *Computermodell* verwendet. Dieser Begriff kann jedoch zu Mißverständnissen führen, da es sich hierbei nicht um ein Modell eines Computers handelt.

[4] Zustandsvariablen sind Modellbeschreibungsgrößen, aus denen sich der Zustand eines Systems vollständig ergibt. Sie sind voneinander unabhängig (eine Zustandsvariable kann nicht als Kombination anderer Zustandsvariablen dargestellt werden) und nicht immer eindeutig definierbar.

zugrundeliegende Modell computerbasiert (und mathematisch) ist – in dieser Kombination bezeichnet man dieses Modell auch als Simulationsmodell. Analog zum Begriff des Modells, wird in dieser Arbeit der Begriff Simulation nur in der Spezialisierung als Computersimulation verwendet. Aus mathematischer Sicht kann man eine Simulation als numerische Lösung von dynamischen (d. h. zeitabhängigen) mathematischen Modellen auffassen.

Die initiale Belegung der Zustandsvariablen des Simulationsmodells wird ausschließlich durch die Werte sogenannter *Modelleingabeparameter* vorgenommen. Das Simulationsmodell beinhaltet entsprechend auch die Definition von Modelleingabeparametern und möglichen Modellausgaben. Diese sind in Form von *Modelleingabeports* und *Modellausgabeports* spezifiziert und ermöglichen den Austausch von typisierten Daten mit der Systemumgebung des Simulationsmodells.

Typischerweise enthält das Simulationsmodell auch eine Wertebelegung für Modelleingabeparameter und definiert, welche Daten als Modellausgaben erzeugt werden. Sowohl die Wertebelegung für Modelleingabeparameter, als auch die Auswahl der Modellausgaben, die an den Modellausgabeports ausgelesen werden, sind jedoch Bestandteile des Experimentes mit dem Modell. Um diese Parameter konzeptionell getrennt zu erfassen, wird in dieser Arbeit der Begriff *Model-Under-Experimentation (MUX)*[5] eingeführt. Er bezeichnet ein Simulationsmodell ohne diese Experiment-spezifischen Anteile. Abbildung 2.2 auf der folgenden Seite visualisiert die Vermischung von Aspekten der Modellierung und des Experimentierens im Simulationsmodell (Teilabbildung 2.2a) und deren konzeptionelle Trennung im MUX (Teilabbildung 2.2b). Dabei beinhaltet das MUX weiterhin die Definitionen für Modelleingabe- und Modellausgabeports. Das MUX abstrahiert zudem von einem System, das einer Domäne zugeordnet werden kann, welche fortan als *MUX-Domäne* bezeichnet werden wird.

2.2. Modellierung und Experimentier-Prozess

Der Begriff *Modellierung & Simulation (M&S)* [ZPK00] bezeichnet einen Prozess, der aus zwei sich bedingenden Teilprozessen besteht: Erstellung eines oder mehrerer Modelle (*Modellierung*) und der Erstellung darauf basierender Modellexperimente bzw. Simulationen (*Experimentier-Prozess*). Abbildung 2.3 auf Seite 25 stellt beide Teilprozesse in Form eines Zyklus dar, der iterativ solange fortgesetzt wird, bis das Untersuchungsziel erreicht wurde (oder andere Umstände zur Einstellung der Experimentserie führen). Das Untersuchungsziel[6] ist hierbei die zentrale Verbindung beider Teilprozesse und die Motivation für jede ihrer Aktivitäten.

[5] Die Bezeichnung *Model-Under-Experimentation (MUX)* ist in Analogie zum Begriff *Model-Under-Test (MUT)* gewählt worden, der im Bereich des modellbasierten Testens verwendet wird.

[6] „Der Experimentator […] sucht durch seine Experimente für diese [ganz bestimmten] Fragen und nur für sie eine Entscheidung zu erzwingen; alle anderen Fragen bemüht er sich dabei auszuschalten." [Pop05, S. 84]

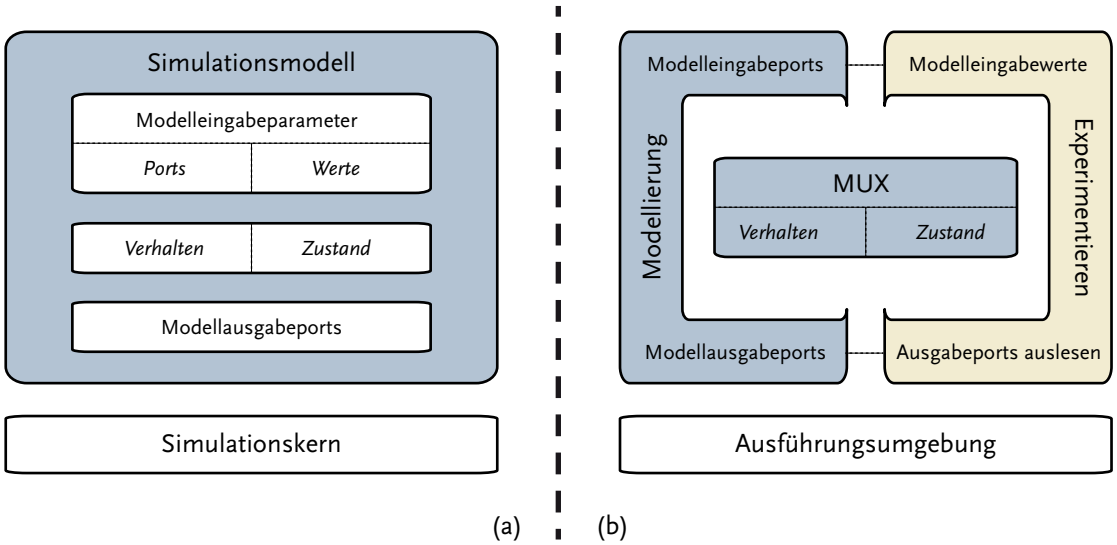


Abbildung 2.2.: Zusammenhang von (a) *Simulationsmodell* und (b) *MUX*

2.2.1. Modellierung

Die ersten beiden Aktivitäten (Beobachtung und mathematische Formalisierung) des Teilprozesses Modellierung wurden bereits während der Klärung des Begriffes Modell hinreichend berücksichtigt (Abschnitt 2.1.3). Die dritte Aktivität *Programmierung* besteht aus folgenden, manuellen Schritten (siehe (a) in Abbildung 2.5 auf Seite 27):

1. Überführung eines formalen, mathematischen Modells in ein computerbasiertes Modell (das MUX bzw. Simulationsmodell),
2. Überführen des computerbasierten Modells in ein ablauffähiges Programm, das als *Simulationsprogramm* (bzw. allgemeiner in dieser Arbeit als *MUX-Programm*) bezeichnet wird. Hierbei müssen mindestens zwei Funktionalitäten hinzugefügt werden: Die Herstellung eines initialen Modellzustandes (durch Belegung aller Zustandsvariablen mit Initialwerten) und die Ansteuerung der Verhaltensmethoden zur Nachbildung des Verhaltens des Originalsystems über die Zeit, was üblicherweise durch einen *Simulationskern* vorgenommen wird,
3. und die Integration des Simulationsprogrammes mit geeigneter Software (i. A. mindestens ein Betriebssystem) und Hardware, was in der Gesamtheit als *Simulator*[7] bezeichnet wird. Er ermöglicht die Ausführung einer Computersimulation.

Der *Simulationskern* ist eine Programmkomponente, die identische Ausführungsalgorithmik für verschiedene Simulationsmodelle als Laufzeitsystem zur Verfügung stellt. Dadurch wird vermieden, in jedem Simulationsmodell redundant dessen Ansteuerung zu implementieren.

[7] Nach Zeigler ist ein Simulator „any computer system [...] capable of executing a model to generate its behavior“ [ZPK00, S. 30].

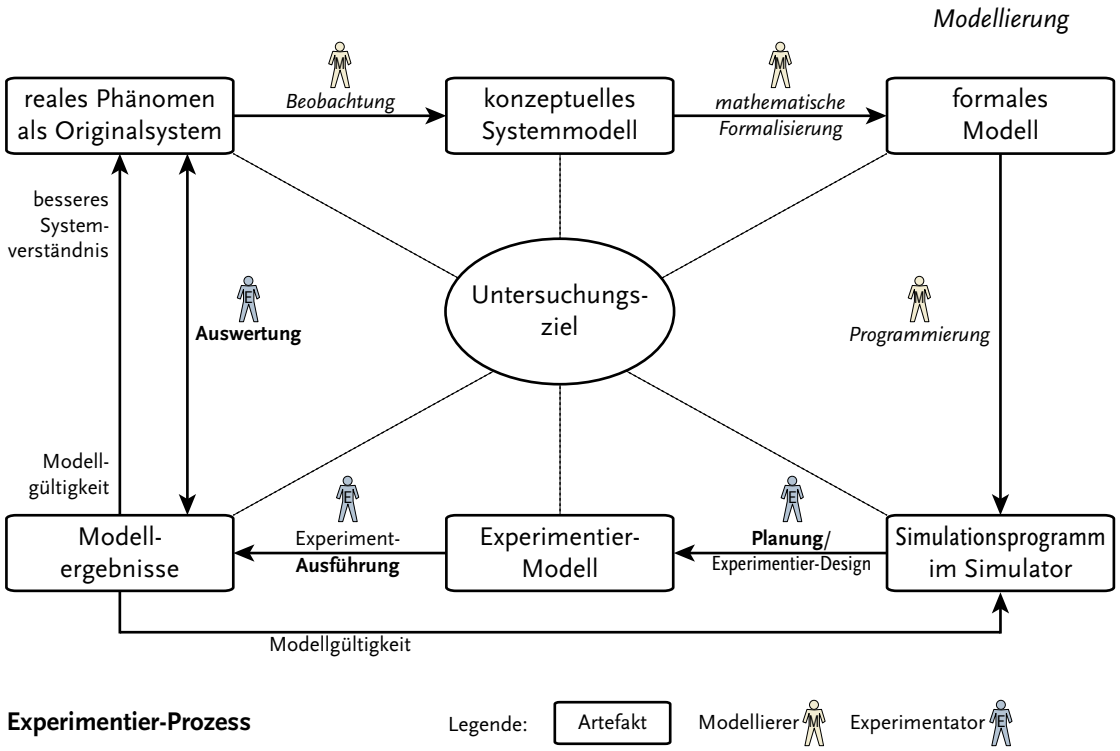


Abbildung 2.3.: Modellierung und Experimentier-Prozess aus Sicht der beteiligten Artefakte (in Anlehnung an [FA96, S. 27])

Der Simulationskern verwaltet typischerweise die *Modellzeit*, also eine abstrakte Repräsentation der Zeit, die im zu analysierenden, originalen System vergeht (*Realzeit*).

Ein Simulationskern bildet auch die Schnittstelle zum Betriebssystem, auf der die Computersimulation ablaufen soll. Entsprechend können die Aufgaben eines bestimmten Simulationskerns variieren. Beispiele hierfür aus der SOSEWIN-Fallstudie sind die beiden alternativen Laufzeitsysteme „Simulationskern des Netzwerksimulators ns-3“ und „C++-Netzwerkbibliotheken und Hardware-Treiber“ auf den Knoten des realweltlichen Prototypnetzwerkes. Beide Laufzeitsysteme stellen einem MUX Netzwerkfunktionen zur Verfügung, jedoch im Fall des Netzwerksimulators auf Basis einer Abstraktionsschicht, die lediglich einen bestimmten Modellzeitverbrauch pro übertragener Nachricht berechnet bzw. simuliert. Hierfür ist die Bezeichnung Simulationskern passend. Bei der Verwendung von Hardware, die in einer realweltlichen Umgebung tatsächlich die Nachrichten überträgt, wird diese Übertragung in Realzeit gemessen. Angesichts dessen scheint die Bezeichnung Simulationskern innerhalb des realweltlichen Prototypnetzwerkes weniger passend, so dass stattdessen in dieser Arbeit die allgemeinere Bezeichnung *Ausführungsumgebung* eingeführt wird. Die Ausführungsumgebung ist ein Laufzeitsystem, das während der Ausführung des MUX-Programmes dessen Steuerung und Beobachtung sowie Managementaufgaben (Speicherallokation, Ein-/Ausgabe, …) übernimmt. Sie wird von *Laufzeitparametern* gesteuert, welche das Verhalten des MUX nicht verändern und damit auch seinen Modellzustand nicht beeinflussen. Laufzeitparameter kön-

nen lediglich die benötigte Zeitdauer für eine Ausführung des MUX beeinflussen (z. B. durch Aktivierung eines Monitoring-Mechanismusses). Somit beeinflussen sie den Experimentlauf, nicht aber das Experiment.

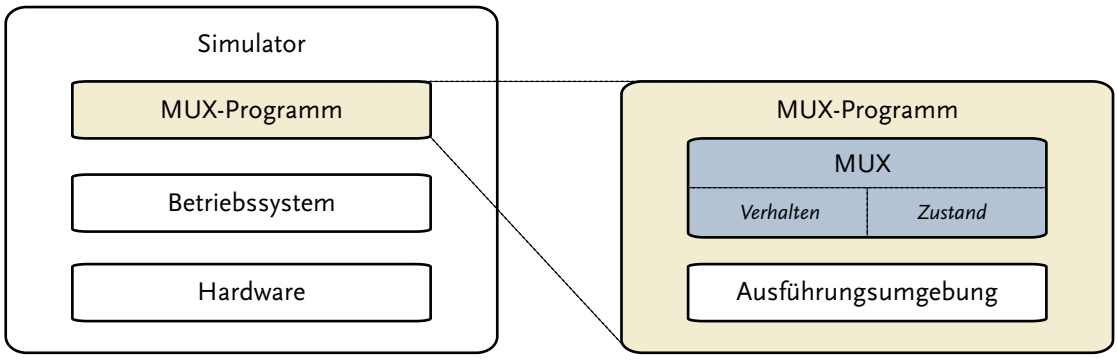


Abbildung 2.4.: Schematischer Aufbau eines Simulators mit MUX, MUX-Programm und Ausführungsumgebung

2.2.2. Experimentier-Prozess

In dieser Arbeit wird der Experimentier-Prozess auf der Basis von Modellen (MUX) untersucht, so dass der bisher dargestellte Prozess der Modellierung als existent vorausgesetzt wird. Der in dieser Arbeit vorgestellte Ansatz kann bereits im Modellierungsprozess an zwei Stellen hilfreich eingesetzt werden: Erstens kann die Ausführung des Compilers automatisiert ausgeführt werden und zweitens kann das Laufzeitsystem automatisiert gesteuert werden. In Kombination mit neueren Verfahren der modellgetriebenen Entwicklung (z. B. MDA) ergeben sich effizientere Möglichkeiten, Modellexperimente mit verschiedenen Untersuchungszielen in unterschiedlichen Ausführungsumgebungen durchzuführen und auszuwerten. Solche Verfahren erlauben es, aus dem MUX, das hierbei als *Platform Independent Model (PIM)* vorliegt, zunächst für die Ausführungsumgebung spezifischen Quellcode – das sogenannte *Platform Specific Model (PSM)* – automatisch zu generieren (was als *Transformation* bezeichnet wird).[8] Das PSM zusammen mit der Ausführungsumgebung entspricht dem MUX-Programm. Somit lassen sich aus einem MUX mehrere MUX-Programme automatisch erzeugen, was gleichzeitig bedeutet, die Ausführungsumgebung wechseln zu können, ohne erneutes, manuelles Erstellen des MUX-Programmes (siehe Teilabbildung (b) in 2.5). Dadurch reduziert sich der Entwicklungsaufwand typischerweise erheblich (da die entsprechenden Transformationswerkzeuge nur einmal entwickelt werden müssen). Dies ist bedeutsam, denn ein Wechsel der Ausführungsumgebung kann das Erreichen eines bestimmten Untersuchungszieles erst ermöglichen oder zumindestens vereinfachen (z. B. können bestimmte Wellenausbreitungseffekte der Realwelt in einem Netzwerksimulator nur unzureichend oder gar nicht abgebildet werden).

[8] PIM und PSM sind Begriffe der *Model-Driven-Architecture (MDA)*, die von der *Object Management Group (OMG)* eingeführt wurden. Eine Einführung hierzu und zu Transformation gibt Abschnitt 9.1.1 auf Seite 119.

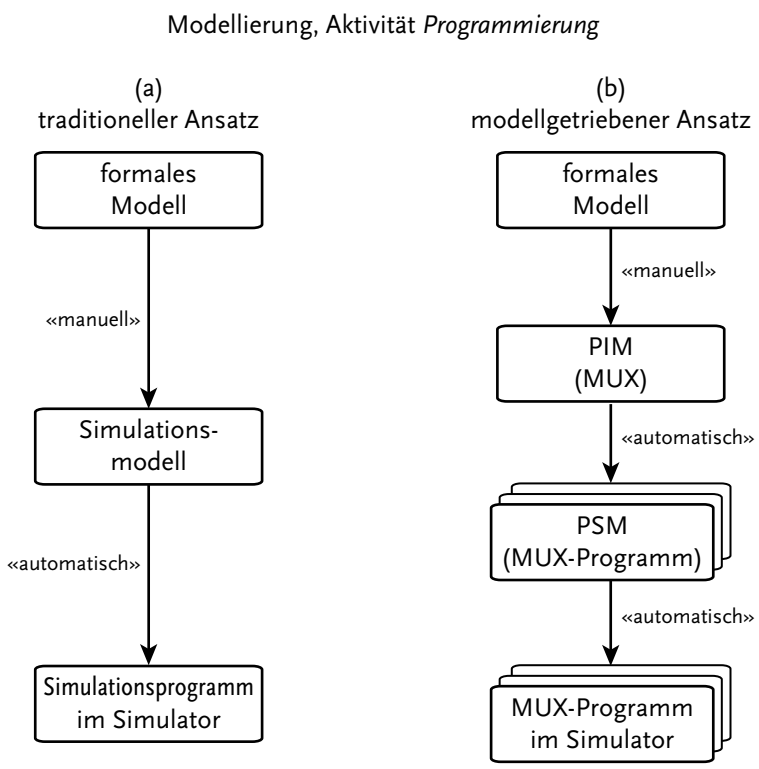


Abbildung 2.5.: Artefakte in der Modellierung im Vergleich: (a) traditioneller Ansatz vs. (b) modellgetriebener Ansatz

Um das Experimentieren mit einem MUX zu erleichtern, soll die initiale Belegung der Zustandsvariablen des MUX ausschließlich abhängig von der Belegung der Modelleingabeparameter erfolgen. Dies schließt insbesondere Variablen ein, die den Pseudozufall innerhalb des Modells steuern (z. B. Initialisierungsvektoren oder auch Verteilungsfunktionen von Zufallsgrößen). Modelleingabeparameter werden erst zu Beginn der Ausführung des MUX-Programmes gesetzt. Dadurch lassen sich Experimentserien durchführen, ohne das einmal erstellte MUX-Programm zu verändern.

Forderungen an MUX und Ausführungsumgebung Zusammengefasst ergeben sich bisher folgende Forderungen (bzw. *best practices*), um die Durchführung und Automatisierung von Modellexperimenten zu erleichtern[9]:

- MUX und Ausführungsumgebung sollen separat existieren und durch definierte Schnittstellen interagieren,
- die initiale Belegung der Zustandsvariablen des MUX ist ausschließlich abhängig von der Belegung seiner Modelleingabeparameter

[9] Hierbei wird das Prinzip der *separation of concerns* angewendet. Dadurch lässt sich beobachtetes Verhalten einfacher bestimmten Teilen zuordnen, als bei einem einzigen, monolithischen Software-Artefakt.

- und das Verhalten der Ausführungsumgebung kann durch Laufzeitparameter beeinflusst werden, ohne aber das Verhalten des MUX zu verändern.

Experimentier-Modell

Der Begriff *Experimentier-Modell* soll an dieser Stelle als abstrakte Beschreibung eines Experimentier-Prozesses verwendet werden. Experimentier-Modelle werden später in dieser Arbeit als Experimentier-Workflows präziser beschrieben (siehe Abschnitt 4.3).[10]

Ein Experimentier-Modell speichert beispielsweise Informationen, die Zeigler in [ZPK00] als *experimental frame* bezeichnet:

> An *experimental frame* is a specification of the conditions under which the system is observed or experimented with. […] One views a frame as a definition of the type of data elements that will go into the database.

Das angesprochene *system* ist hierbei das MUX in Verbindung mit dessen Ausführungsumgebung. Zu den *data elements* im Experimentier-Modell gehören mindestens, damit eine Ausführung des MUX-Programmes möglich wird:

- die initiale Belegung der Zustandsvariablen des MUX und
- die Laufzeitparameter der Ausführungsumgebung.

Das Experimentier-Modell soll im Sinne einer guten wissenschaftlichen Praxis[11] zudem die Reproduzierbarkeit des Experimentier-Prozesses mit gleichen Ergebnissen ermöglichen (siehe auch Abschnitt 7.4.1), so dass darin zusätzlich enthalten sein müssen:

- persistente Referenzen auf alle im Experimentier-Prozess beteiligten Artefakte (vor allem MUX, MUX-Programm und Ausführungsumgebung) und
- sämtliche im Experimentier-Prozess durchgeführten Aktivitäten, deren Abfolge und der zwischen ihnen stattfindende Datenfluss.

Andernfalls können Modellergebnisse nicht in Relation zu dem Experiment gesetzt werden, das sie erzeugt hat. Das ist jedoch eine Voraussetzung, damit sie sinnvoll interpretiert werden können.

---

[10] Die Relation des Experimentier-Modells zur Workflow-Terminologie ist in Abschnitt 5.4 auf Seite 55 beschrieben und in Abschnitt 7.4 auf Seite 89 werden Anforderungen an die Beschreibung von Experimentier-Prozessen detailliert betrachtet.

[11] „Alle Resultate eines Experimentes oder einer Studie zu dokumentieren, und die Primärdaten zu sichern und aufzubewahren“ wird als eines der allgemeinen Prinzipien guter, wissenschaftlicher Praxis an der Humboldt-Universität zu Berlin angesehen [Ber02]. Zu den erwähnten Primärdaten gehören nach Meinung des Autors der vorliegenden Arbeit insbesondere Experimentier-Prozesse.

2.3. Experimentelle Untersuchung in der Systemanalyse und Modellklassen

Die Systemanalyse hat den Zweck, das Verständnis für das Verhalten realer Phänomene zu verbessern, um eine Entscheidungsbasis für den Umgang mit diesen Phänomenen zu erhalten. Solche Phänomene als Gegenstand von experimentellen Untersuchungen können sehr vielfältiger Natur sein: Sie können der objektiven Realität entstammen oder auch imaginäre Konstrukte sein, die möglicherweise hypothetische Phänomene der Realität betrachten. Wie bereits dargelegt, fasst man diese Phänomene als Systeme auf und bedient sich für ihre Untersuchung der Technik der Modellierung. In dieser Arbeit sind die dabei erstellten Modelle von mathematischer, numerischer Art. Diese Modelle werden experimentell untersucht, indem man das Verfahren der Computersimulation nutzt. Dabei spielen vor allem *dynamische*[12] (zeitbehaftete) Modelle eine Rolle. Diese Modellklasse ist hauptsächlich Gegenstand der Experimentier-Prozesse, die in dieser Arbeit untersucht werden (schattiert hervorgehoben in Abbildung 2.6). Allerdings wäre der gewählte Ansatz auch für statische (nicht-zeitbehaftete) Modelle geeignet.

2.3.1. Modellkopplung

Das Untersuchungsziel im Experimentier-Prozess kann formuliert sein als einfaches, manuelles Auswerten der Modellergebnisse bis hin zu komplexen Aufgabenstellungen (wie z. B. Polyoptimierung). Hierbei kann es vorkommen, dass Aspekte des zu untersuchenden Systems auf mehrere Modelle aufgeteilt werden, um beispielsweise diese Einzelmodelle zunächst separat zu untersuchen. Das übergeordnete Untersuchungsziel erfordert jedoch ein Zusammenspiel dieser Einzelmodelle, was als *Modellkopplung* bezeichnet wird. Modellkopplungen können entweder sequentiell oder parallel sein. Im sequentiellen Fall ist die Ausführung des einen Modells beendet, bevor die Ausführung des nachfolgenden Modells beginnt. Somit kann das nachfolgende Modell bei seiner Ausführung auf die Modellbeobachtungen des ersten Modells zugreifen. Im parallelen Fall laufen die beteiligten Einzelmodelle nebenläufig ab und können sich zu festgelegten Zeitpunkten synchronisieren, d. h. die Belegung ihrer Zustandsvariablen abfragen, woraufhin sich das eigene Modellverhalten entsprechend verändern kann. Der in dieser Arbeit entwickelte Ansatz unterstützt die sequentielle Modellkopplung. Die parallele Modellkopplung wird derzeit nicht unterstützt.

[12] Dynamische Modelle werden weiter differenziert in zeitdiskrete und zeitkontinuierliche Modelle. Verändert sich der Systemzustand nur zu bestimmten, ausgezeichneten Ereigniszeitpunkten, so spricht man von zeitdiskreter Simulation. Ist das Modell durch Differentialgleichungen beschrieben, spricht man von zeitkontinuierlicher Simulation. Beide Arten können ereignisorientiert und/oder prozessorientiert modelliert werden. Für weitere Details wird auf die Literatur verwiesen, z. B. [Fis82, FA96].

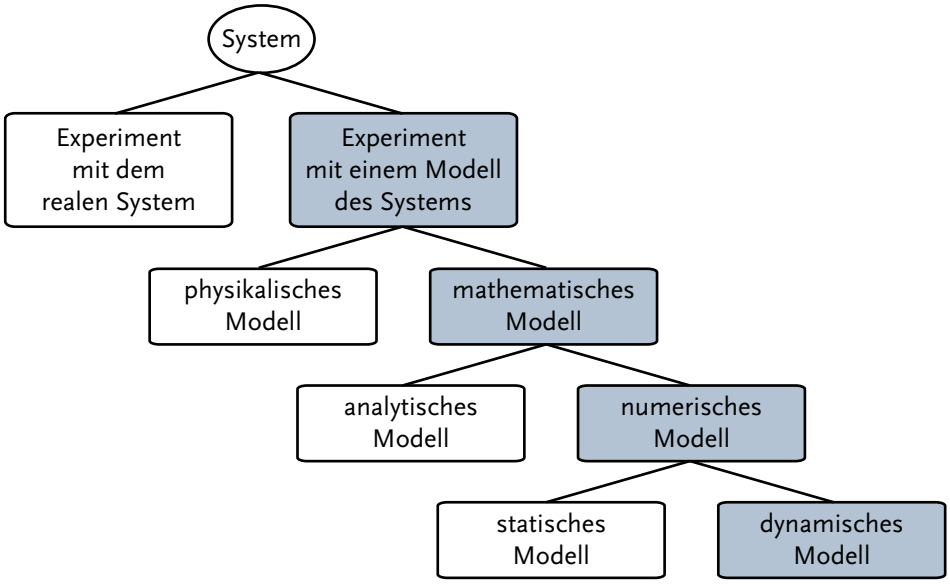


Abbildung 2.6.: Möglichkeiten, ein System zu untersuchen sowie Modellklassen

2.3.2. Beispiel aus der Literatur

An einem Beispiel aus der Literatur sollen die zuvor definierten Begriffe verdeutlicht werden. Hierzu wird das Beispiel von Hartmut Bossel „Dynamik des Fischfangs“ [Bos94, S. 127 ff.] verwendet, auf das in folgenden Kapiteln unter der verkürzten Bezeichnung „Bossel-Fischfang“ Bezug genommen wird.

Das System im Bossel-Fischfang ist dynamisch und hat die Struktur eines klassischen Räuber-Beute-Systems von Lotka und Volterra mit logistischer Sättigung bis zur Kapazitätsgrenze [Bos94, S. 288 f.]. Dabei existiert eine dynamische Interaktion zwischen einer Räuberpopulation (Fischerboote) und einer logistisch wachsenden Beutepopulation (Fische), deren maximaler Bestand durch eine Kapazitätsgrenze beschränkt ist (Nährstoffangebot für Fische). Die Beutemenge ist sowohl proportional zur Beutepopulation als auch zur Räuberpopulation. Ein Zuwachs in der Räuberpopulation führt zu Verlusten in der Beutepopulation. Unterschreitet die Beutepopulation ein kritisches Maß, so können nicht mehr alle Räuber ernährt werden, und ihre Zahl nimmt ab (im Bossel-Fischfang wird dadurch das Betreiben von zuviel Fischerbooten wirtschaftlich unrentabel). Weniger Räuber bedeutet auch weniger Beutemenge, und die Beutepopulation wird sich wieder erholen. Dieses Verhalten führt zu sich periodisch wiederholendem Schwingen beider Populationen, wobei sich das System unabhängig von den Anfangsbedingungen auf einen stabilen Gleichgewichtspunkt einschwingt.

Das Untersuchungsziel bzw. Modellzweck soll eine Entscheidungshilfe sein, wann unter welchen ökologischen und ökonomischen Bedingungen wieviele Boote sinnvoll eingesetzt werden sollen, um eine maximale Beutemenge zu erreichen. Auf der Basis von Untersuchungsziel und Systembeschreibung (die in [Bos94, S. 127 ff.] noch weiter verfeinert wird) kann die Modellbildung beginnen. Dabei wird zunächst ein mathematisches Modell in Form zwei-

er Differentialgleichungen erstellt, wobei jeweils eine die Entwicklung der Zustandsgrößen *Fischpopulation* und der *Anzahl der Boote* über die Modellzeit beschreibt [Bos94, S. 133]. Dieses mathematische Modell ließe sich bereits analytisch lösen, ohne die Methode der Simulation anzuwenden. Allerdings sind Systeme und Fragestellungen, die üblicherweise simulativ untersucht werden, zu komplex, um ein einfaches Beispiel zu bieten.

Aus diesem mathematischen Modell wird ein Simulationsmodell erstellt [Bos94, S. 169 ff.]. Dieses Simulationsmodell ist zudem ein Beispiel für eine Vermengung mit Experiment-spezifischen Parametern. Es enthält beispielsweise Wertebelegungen für Modelleingabeparameter und Auswertungsroutinen, die Modellausgaben aggregieren und aufbereiten. Dennoch soll vereinfachend angenommen werden, dieses Simulationsmodell sei das MUX in diesem Beispiel, obwohl hierfür die Experiment-spezifischen Parameter entfernt und in das Experimentier-Modell ausgelagert werden müssten. Das MUX wird mit Hilfe der Computersprache *Simulation of Processes and Systems (SIMPAS)* [Bry80] implementiert und stellt zusammen mit der zugehörigen Ausführungsumgebung das MUX-Programm dar. SIMPAS basiert auf TurboPascal, das in diesem Fall die Ausführungsumgebung ist. Teile des Simulators werden auf der Basis des MUX-Programmes durch den Compiler von TurboPascal erzeugt.

3. Simulationssysteme

Ein *Simulationssystem* ist ein Softwaresystem und stellt eine Werkzeugsammlung dar, mit der Modellierung & Simulation praktiziert werden kann. Ein Simulationssystem besteht nach [Fis82] aus:

- der Definition einer Computersprache, mit der das Simulationsmodell formal notiert ist (eine solche Sprache wird als *Simulationssprache* bezeichnet),
- einem Compiler, der das Simulationsmodell in ausführbare Programmteile überführt (woraus sich das Simulationsprogramm bilden lässt) bzw. ein Interpreter, der das Simulationsmodell direkt ausführen kann,
- einem Laufzeitsystem (die Ausführungsumgebung) zur Steuerung und Beobachtung des Simulationsmodells während seiner Ausführung, das auch Managementaufgaben (Speicherallokation, Ein-/Ausgabe, …) übernimmt, und
- vorimplementierten Standardalgorithmen, um die Modellentwicklung zu vereinfachen (z. B. statistische Hilfsfunktionen, Zufallszahlengeneratoren, …).

Simulationssysteme existieren bereits, solange M&S betrieben wird. Sie sind unterschiedlich ausgerichtet bezüglich unterstützer Formalismen (z. B. DEVS, DESS), Modellierungsparadigmen (ereignis- bzw. prozessorientiert), Computersprachen, Laufzeitsystemen und dem Umfang angebotener Standardalgorithmen. Deshalb existiert eine Fülle von unterschiedlichen Simulationssystemen, die jeweils Stärken für die Modellierung in verschiedenen Domänen mit sich bringen.

Simulationssysteme bieten üblicherweise Funktionen für die Datenerfassung innerhalb des Simulationsmodells, um Modellbeobachtungen ausgeben zu können. Dabei lassen sich grundsätzlich Mechanismen unterscheiden, die automatisch Daten erfassen, indem sie mit bestimmten Konzepten im Simulationskern verbunden sind, und solchen, die explizit im Programmcode aufgerufen werden müssen. Die so erfassten Daten können zudem bereits innerhalb des Simulationsmodells ausgewertet werden, beispielsweise mittels statistischer Funktionen. Die unausgewerteten Rohdaten sind bei einem solchen Vorgehen jedoch verloren, wodurch eine andere Auswertung später nicht mehr möglich ist. Somit ist auch die Beantwortung anderer Fragestellungen nicht möglich, obwohl die Rohdaten dies möglicherweise zugelassen hätten. Eine Trennung von Systemmodellierung und Experimentieren mit dem Systemmodell würde in diesem Fall bedeuten, die Rohdaten als Modellbeobachtung auszugeben und als Experimentdaten zu speichern. Sofern keine gewichtigen Gründe dagegen sprechen (z. B. eine zu große Datenmenge), ist dieses Verfahren dem Auswerten im Simulationsmodell vorzuziehen.

Die Gebundenheit an eine spezifische Simulationssprache ist charakteristisch für Simulationssysteme. Modellbasierte Ansätze sind selten, bei denen das Simulationsmodell in einer

Computersprache vorliegt, die zunächst unabhängig vom verwendeten Simulationssystem ist, woraus jedoch ein Simulationsprogramm erzeugt werden kann, das spezifisch für ein Simulationssystem ist.[1] Der Wechsel von einem Simulationssystem zu einem anderen bedeutet somit einen erheblichen Aufwand, bei dem das Simulationsprogramm neu erstellt werden muss. Ein solcher Wechsel wäre hilfreich, um jeweils spezifische Fragestellungen beantworten zu können, die nur mit einem anderen Simulationssystem geklärt werden können.

Eine *Simulationsbibliothek* ist ein Spezialfall eines Simulationssystems, bei dem

- die Simulationssprache identisch zu einer existierenden Programmiersprache bzw. einer General-Purpose-Language (z. B. C++ oder Java) ist,[2]
- das Laufzeitsystem und die angebotenen Standardalgorithmen (z. B. in einem Simulationskern) ebenfalls in dieser Programmiersprache implementiert sind und
- die entsprechenden Simulationsbinärprogramme direkt mit einem Standardcompiler der zugrundeliegenden Programmiersprache erzeugt werden können (und das Simulationssystem somit keinen separaten Compiler bereitstellen muss).

In diesem Kapitel werden Simulationssysteme hinsichtlich ihrer Eignung zur Unterstützung des Experimentier-Prozesses betrachtet. Dabei wird überprüft, ob folgende, aufeinander aufbauende Experimentierkonzepte in dem jeweiligen Simulationssystem vorhanden sind:

Experimentkonzept Die Parameter für das Experiment sind getrennt von den Modelleingabeparametern des Simulationsmodells (und für die Laufzeitumgebung). Zu den Experimentparametern gehören beispielsweise Wertebelegungen für die Modelleingabeparameter (z. B. Angabe der Modellzeitdauer), welche Modellbeobachtungen erfasst werden sollen und eine Beschreibung des Untersuchungszieles.

Experimentserien Hierbei können mehrere Experimente beschrieben werden, die (sequentiell oder parallel) in einem Arbeitsablauf ausgeführt werden. Dies setzt ein Experimentkonzept voraus.

Parameterbelegungsvariationen Die Belegung der Modelleingabeparameter wird anhand einer bestimmten Vorschrift variiert, wodurch sich mehrere Experimente in einer Experimentserie ergeben (die Konzepte Experimentserie und Experimentkonzept sind eingeschlossen).

Darauf aufbauend werden Simulationssysteme in dieser Arbeit in drei Klassen unterteilt:

1. *Simulationssysteme ohne Experimentierunterstützung* verfügen über keines der genannten Experimentierkonzepte.
2. *Simulationssysteme mit externer Experimentierunterstützung* verfügen über mindestens eines der Experimentierkonzepte, benötigen hierfür aber ein separates Softwaresystem.

[1] Ein Beispiel für einen solchen, modellbasierten Ansatz ist das *Real Time Developer Studio (RTDS)* von *PragmaDev* [20009f], das in der SOSEWIN-Fallstudie eingesetzt wird (Abschnitt 7.3.1).

[2] Die Simulationssprache darf programmiersprachenkonforme Definitionen von Klassen, Methoden, u. ä. enthalten. Definierte sie aber eine eigene Syntax und Semantik auf Basis der zugrundeliegenden Programmiersprache, so wäre dies eine neue Sprache, eine sogenannte *interne DSL* (siehe Kapitel 6 auf Seite 59).

3. *Simulationssysteme mit interner Experimentierunterstützung* verfügen über mindestens eines der Experimentierkonzepte (wobei die Funktionalität hierfür integriert ist).

In den folgenden Abschnitten werden Eigenschaften und Beispiele von Simulationssystemen dieser Kategorien vorgestellt.

3.1. Simulationssysteme ohne Experimentierunterstützung

Simulationssysteme ohne Experimentierunterstützung sind sehr verbreitet. Ihr Fokus liegt auf der Modellierung und der Erstellung von Simulationsprogrammen. Konzeptionell unterscheiden diese Systeme nicht zwischen der Ausführung des Simulationsprogramms und eines Experimentes mit dem Simulationsprogramm. Das Laufzeitsystem eines Simulationssystems ohne Experimentierunterstützung führt ein Simulationsprogramm typischerweise einmal aus und beendet sich anschließend selbst. Es ist nicht in der Lage, den Zustand des Simulationsprogramms neu- bzw. zurückzusetzen, um es erneut ausführen zu können. Eine neuerliche Ausführung des Simulationsprogramms bedeutet auch eine erneute Ausführung des Laufzeitsystems.

Der praktische Einsatz von Simulationssystemen ohne Experimentierunterstützung verlangt vom Experimentator, sich mit dem Management seiner Experimente selbst auseinanderzusetzen. Sofern seine Experimente eine gewisse, kritische Anzahl erreicht haben, muss er sich ein – wie auch immer geartetes – System erarbeiten und aufbauen, mit dem er die Übersicht behält. Weiß der Experimentator nicht mehr, durch welchen Experimentier-Prozess und durch welche Artefakte ein Ergebnis entstanden ist, verliert es seinen Wert. Exemplarisch werden zu den im Folgenden vorgestellten Simulationssystemen auch reale Einsatzszenarien vorgestellt, bei denen immer eine Experimentierunterstützung manuell hinzugefügt wurde.

Simulationssysteme ohne Experimentierunterstützung profitierten deshalb stark von einer potentiellen Kombination mit dem in dieser Arbeit vorgestellten Ansatz. Hierfür müssen nur geringe Voraussetzungen bezüglich der Modellein- und Ausgabeports des MUX und der Ausführungsumgebung erfüllt werden (auf die in 2.2.2 bereits hingewiesen wurde). Außer der Erfüllung dieser Voraussetzungen sind keine (weiteren) Änderungen am Simulationssystem nötig.

3.1.1. ODEM und ODEMx

ODEM (*Object-oriented-Discrete-Event-Modelling*) ist eine Simulationsbibliothek, die am Lehrstuhl *Systemanalyse* des Instituts für Informatik der Humboldt-Universität zu Berlin entwickelt wurde [FA96]. Sie unterstützt zeitdiskrete und zeitkontinuierliche Simulationsmodelle, die in C++ implementiert werden müssen. Der Simulationskern liegt ebenfalls in C++ vor. Eine Experimentierunterstützung bietet ODEM nicht. In *ODEMx* wurden die Konzepte von ODEM weiter entwickelt und verfeinert. ODEMx entstandt im Rahmen einer Diplomarbeit [Ger03], wobei die Softwarearchitektur von ODEM unter Verwendung neuer Konzepte der Programmiersprache C++ überarbeitet wurde.

In ODEMx gibt es drei grundlegende Konzepte für die Erfassung von Modellbeobachtungsdaten innerhalb des Simulationsprogramms: `Trace`, `Observation` und `Report`. Der Trace-Mechanismus erlaubt die Ausgabe selbstdefinierter Meldungen an beliebigen Stellen innerhalb des Simulationsprogramms. Dabei kann einer Meldung eine bestimmte Trace-Kategorie zugeordnet werden, was beispielsweise späteres Filtern nach bestimmten Kategorien erlaubt. Das Observation-Konzept dient der Datenerfassung aus bestimmten Objekten, die Instanzen von Klassen des Simulationsprogramms sind. Dabei müssen diese Objekte manuell bei anderen Objekten zur Überwachung registriert werden. Einzig der Report-Mechanismus bietet einen Automatismus, der automatisch Statistikberichte auf der Basis aggregierter Daten für Objekte vordefinierter Klassen erstellt.

ODEMx wurde in der SOSEWIN-Fallstudie eingesetzt (siehe Abschnitt 7.3.1). Nicht zuletzt durch die fehlende Experimentierunterstützung von ODEMx und aus projektbezogenen Gründen musste in der Fallstudie ein eigenständiges Experiment-Management-System entwickelt werden. In der SOSEWIN-Fallstudie wird auch gezeigt, wie ODEMx konzeptionell mit dem in dieser Arbeit vorgestellten Ansatz kombinierbar ist, ohne Veränderungen an der ODEMx-Bibliothek vornehmen zu müssen (siehe Abschnitt 10.3.1).

3.1.2. JiST/SWANS

JiST/SWANS ist eine Kombination der auf Java basierenden Simulationsbibliothek *Java in Simulation Time (JiST)* [Bar04] und dem darauf aufbauenden *Scalable Wireless Ad-hoc Network Simulator (SWANS)* [BHVR05]. JiST ist eine eigenständige Simulationsbibliothek, die in der Praxis jedoch vergleichsweise selten zur Implementierung von Simulationen verwendet wird. In Kombination mit dem ebenfalls an der Cornell University entstandenen Netzwerksimulator SWANS erfreut sie sich in der entsprechenden Community jedoch einiger Beliebtheit.

JiST/SWANS wird beispielsweise eingesetzt, um Netzwerksimulationen für das *Humboldt-Wireless-Lab (HWL)* [SZS12b, HWL11] vorzubereiten. Beispielsweise werden veränderte Implementierungen der Media-Access-Control-Schicht zunächst mit JiST/SWANS simuliert, bevor sie auf dem HWL unter weniger abstrakten Bedingungen getestet werden. JiST/SWANS bietet keine Experimentierunterstützung, weshalb eine entsprechende Infrastruktur namens *DistSim* aufgebaut wurde [4]. Sie basiert auf Java und einer zentralen, relationalen MySQL-Datenbank, in der Simulationskonfigurationen und Simulationsergebnisse gespeichert werden. DistSim stellt ein einfaches Scheduling bereit, bei dem auf Maschinen, die Simulationen ausführen sollen, ein Scheduler läuft, der die nächste, in der zentralen Datenbank eingetragene Simulation startet. DistSim ist ein internes Werkzeug der Arbeitsgruppe und wird nicht offiziell als Erweiterung von JiST/SWANS beworben (weshalb JiST/SWANS hier auch als Simulationssystem ohne Experimentierunterstützung geführt wird).

3.1.3. SIMPAS

Das Simulationsmodell im Bossel-Fischfang-Beispiel ist mit Hilfe der Sprache *Simulation of Processes and Systems (SIMPAS)* implementiert worden [Bry80]. SIMPAS unterstützt ereignis-

orientierte Simulationsmodelle, die zeitkontinuierlich oder zeitdiskret sein können. SIMPAS-Programme werden mittels eines bereitgestellten Präprozessors in Pascal-Programme überführt und sind somit mit einem Standard-Pascal-Compiler in ablauffähigen Maschinencode übersetzbar. SIMPAS bietet keine Experimentierunterstützung. Im Bossel-Fischfang-Beispiel wird die graphische Oberfläche von TurboPascal verwendet, um eine manuelle Auswertung von Diagrammen zu ermöglichen, welche die Simulationsergebnisse visualisieren.

3.2. Simulationssysteme mit Experimentierunterstützung

Simulationssysteme mit Experimentierunterstützung bieten eingeschränktes Management von Modellexperimenten hinsichtlich Planung und Ausführung. Eine Auswertung wird, wie für Simulationssysteme typisch, bereits üblicherweise innerhalb des Simulationsmodells definiert. Das Laufzeitsystem eines Simulationssystems mit Experimentierunterstützung führt ein Simulationsprogramm typischerweise mehrfach aus, bevor es sich anschließend selbst beendet. Es ist in der Lage, den Zustand des Simulationsprogramms neu- bzw. zurückzusetzen, um es erneut ausführen zu können.

3.2.1. SLX

Simulation Language with Extensibility (SLX) ist ein Simulationssystem, das in C++ implementiert wurde [Hen95]. *Extensibility* bezieht sich dabei vor allem auf ein Spracherweiterungskonzept, mit dem Datentypen, Anweisungen und Makros definiert werden können. Hierfür werden bereits in SLX vorhandene Basissprachelemente verwendet. SLX unterstützt ereignisorientierte Simulationsmodelle, die sowohl zeitdiskrete als auch zeitkontinuierliche Anteile enthalten können. Eine Besonderheit von SLX ist die pseudo-parallele Ausführung von entsprechend gekennzeichneten Teilen eines Simulationsmodells. Dieses Konzept kann man als eine Übertragung des Unix-Fork[3] in den Simulationskontext betrachten. Eine Anwendung dieser Konzepte im Barbershop-Beispiel könnte beispielsweise die Einführung einer neuen Station sein, in der Kunden ihre Haare von einer Haartrocknungsmaschine getrocknet bekommen. Dieser Teilprozess könnte parallel zur weiteren Arbeit von Joe laufen und die Haartrocknung könnte zeitkontinuierlich beschrieben sein.

Bereits die SLX-Vorgänger, GPSS und GPSS/H, sind in der Lage, den Initialzustand des Simulationsmodells zurück zu setzen, um es erneut ausführen zu können. Die einmalige Ausführung des Simulationsmodells innerhalb von SLX entspricht einem Experimentlauf bzw. einer Stichprobe aus statistischer Sicht. SLX ist in der Lage, die Berechnung statistischer Parameter (z. B. Mittelwert, Konfidenzintervall) für die Ergebnisgrößen aller Stichproben vorzunehmen. Das Erreichen eines bestimmten Konfidenzintervalls für einen interessierenden Parameter kann beispielsweise zu einem Untersuchungsziel in einem Experiment gehören,

[3] Der Unix-Systemaufruf *Fork* erzeugt eine Kopie des aktuellen Prozesses (der das Kommando aufruft), welche dann als Kindprozess des erzeugenden Unix-Prozesses läuft. Der Kindprozess übernimmt dabei die Daten des Elternprozesses.

wodurch das Konfidenzintervall zu einem Bestandteil der Experimentparameter wird. SLX erlaubt somit die Spezifikation von Experimentserien, jedoch keine Parameterbelegungsvariationen. Abbildung 3.1 visualisiert die Ausführung eines solchen Experimentier-Prozesses in SLX.

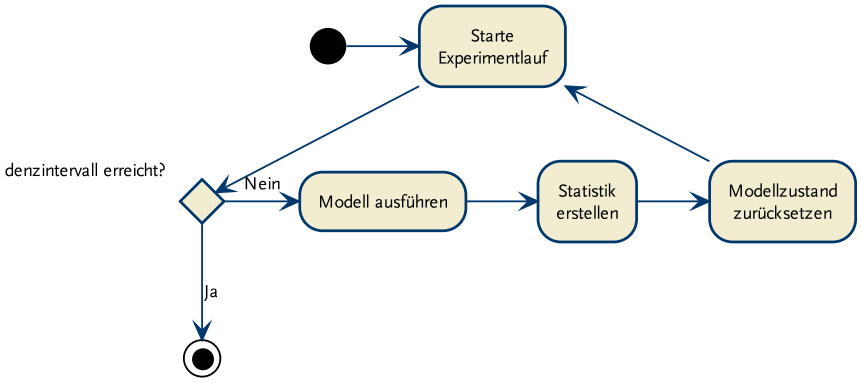


Abbildung 3.1.: Beispiel für die Ausführung eines Experimentier-Prozesses in SLX (als UML-Aktivitätsdiagramm)

3.2.2. DESMO-J mit DISMO

Mit der Simulationsbibliothek *Discrete Event Simulation and MOdelling in Java (DESMO-J)* können diskrete Simulationen implementiert werden, die sowohl ereignisorientierte als auch prozessorientierte Simulationsmodelle umfassen können [PLC00]. *Distributed-Simulation-Optimisation (DISMO)* setzt darauf auf und vereint drei Funktionalitäten [GP01]:

- die Erstellung von Simulationsbinärprogrammen auf Basis der Simulationsbibliothek DESMO-J,
- die verteilte und parallele Ausführung dieser Simulationsbinärprogramme auf verschiedenen Computern und
- Optimierungsmethoden, die auf der Basis von Zielfunktionen arbeiten (in [GP01] werden nur genetische Algorithmen genannt).

DISMO verfügt über ein Experimentkonzept, so dass Parameter des Simulationsmodells von denen des Experiments getrennt sind. Parameterbelegungsvariationen werden in DISMO mit Hilfe von genetischen Algorithmen [SHF94] berechnet, die mit einer vom Experimentator spezifizierten Zielfunktion arbeiten. Mit diesem Mechanismus lassen sich Experimentserien automatisch erzeugen. Jedes Experiment einer solchen Serie kann von DISMO verteilt und parallel auf verschiedenen Computern ausgeführt werden, wobei ein zentraler Server die Steuerung übernimmt. Die Experimente werden mittels *Remote Method Invocation (RMI)* ausgeführt. DISMO stellt an die verarbeitbaren Simulationsmodelle die gleichen Anforderungen, die auch auf Seite 27 als *Forderungen an MUX und Ausführungsumgebung* aufgestellt wurden.

4. Definition und Arten von Workflows

In diesem Kapitel 4 werden grundlegende Begriffe zur Modellierung von Abläufen (bzw. Prozessen) in unterschiedlichen Domänen eingeführt. Dabei wird zunächst die Basisterminologie anhand der historisch ältesten und am besten untersuchten Domäne der Geschäftsprozessmodellierung vorgestellt. Auf dieser Terminologie aufbauend werden wissenschaftliche- und Experimentier-Workflows definiert.

4.1. Business-Workflows

Die *Workflow Management Coalition (WfMC)* ist eine weltweite, 1993 gegründete Organisation von Anwendern, Entwicklern und Forschern, die sich im Bereich Workflows und Geschäftsprozessmodellierung – *Business-Process-Modeling (BPM)* – engagieren. Entsprechend konzentriert sich die WfMC auf den *Business-Workflow (B-Wf)*, doch die zugrundeliegende Terminologie ist unabhängig von der Ausrichtung auf Geschäftsprozesse und findet in dieser Arbeit Anwendung [WfM99].[1] Die Basis für den Aufbau der Workflow-Terminologie ist ein (Geschäfts-)Prozess.

Prozess

Ein *Prozess* ist, sehr allgemein ausgedrückt, eine Sicht auf realweltlich ablaufende, geordnete Schritte. Davenport definiert den Begriff *Geschäftsprozess* wie folgt [SF03, S. 47]:

> Simply a structured, measured set of activities designed to produce a specific output for a particular customer or market. It implies a strong emphasis upon *how* work is done within an enterprise, in contrast to a product focus's emphasis on *what*. A process is thus a specific ordering of work activities across time and place, with a beginning, and end, and clearly identified inputs and outputs: a structure for an action.

Das ist allgemein übertragbar auf einen Prozess, der demnach zeitlich und räumlich strukturiert ist, sowie ein spezifisches Ziel hat. Ein Prozess ist definiert in einer *Prozessdefinition*, die eine Repräsentation (und gewöhnlich auch eine Abstraktion) dessen ist, was beabsichtigt ablaufen soll. Die Prozessdefinition kann in Teilprozessdefinitionen strukturiert werden.

[1] Die Übersetzung der englischen Terminologie ins Deutsche folgt einem Verzeichnis, das der WfMC zur Diskussion vorliegt [Kre99].

Aktivität

Eine Prozessdefinition definiert einen realweltlichen Prozess und ist zusammengesetzt aus *Aktivitäten*. Eine *Aktivität*[2] ist eine Beschreibung einer logischen Arbeitseinheit. Sie stellt typischerweise die kleinste Arbeitseinheit dar, die von einem *Workflow-Management-System (WfMS)* verwaltet wird.[3] Eine Aktivität besteht wiederum aus atomaren *Elementaraufgaben* (*work items*). Aktivitäten werden entweder als Teil eines Workflow-Systems automatisiert (*Automatisierte Aktivität*) oder lediglich modelliert, ohne eine Automatisierung anzustreben (*Manuelle Aktivität*). Die Modellierung von manuellen Aktivitäten innerhalb einer Prozessdefinition dient vor allem der Übersicht, der Vollständigkeit und fördert das Verständnis des modellierten, realen Prozesses.

Eine Prozessdefinition, die ausschließlich aus automatisierten Aktivitäten komponiert ist, wird als *Workflow-Definition* bezeichnet. In der Literatur wird häufig das Wort „Definition“ eingespart und synonym von *Workflow* oder *Workflow-Modell* gesprochen. Ein WfMS als weiterer Teil eines Workflow-Systems kontrolliert die automatisierten Aspekte eines Prozesses. Das WfMS erzeugt und managt *Workflow-Instanzen*, basierend auf den entsprechenden Workflow-Definitionen. Workflow-Instanzen wiederum beinhalten *Aktivitätsinstanzen*.

Ressource

Eine *Ressource* (frz. *la ressource* „Mittel, Quelle“ von lat. *resurgere* „hervorquellen“ [Dud11]; auf dt. „Betriebsmittel“ [Kre99]) ist allgemein ein Mittel, um eine Handlung zu tätigen oder einen Vorgang ablaufen zu lassen. Im Kontext von Workflows erfordert z. B. eine Aktivität für ihre Ausführung Ressourcen in Form von Menschen (wie beispielsweise in Form eines Bearbeiters, siehe nachfolgender Absatz) oder Maschinen (wie beispielsweise in Form eines Computersystems).

Bearbeiter

Aktivitäten, die menschliche Ressourcen benötigen, werden an *Bearbeiter* zugewiesen. Ein Bearbeiter (*workflow participant*) ist eine Ressource, welche die durch eine Aktivitätsinstanz (in Elementaraufgaben) definierte Arbeit durchführt. Der Begriff Bearbeiter wird normalerweise auf eine menschliche Ressource angewendet, kann konzeptionell betrachtet aber auch auf maschinelle Ressourcen angewendet werden.

Abbildung 4.1 fasst die bisher dargestellte, grundlegende Workflow-Terminologie grafisch zusammen. Das in Abbildung 4.1 ebenfalls dargestellte Workflow-Management-System kontrolliert u. a. den Lebenszyklus des Workflows. Unter dem Begriff *Workflow-Lebenszyklus* sind Aktionen zusammengefasst, die für Definition, Instanziierung und Ausführung von Workflow-Instanzen nötig sind.

[2] Synonyme Begriffe sind: *Task*, *Schritt*, *Operation*, *Instruktion*, *work element* und *process element*.

[3] Die Begriffe *Workflow-Management-System* und Workflow-System werden in Kapitel 5 detailliert erläutert.

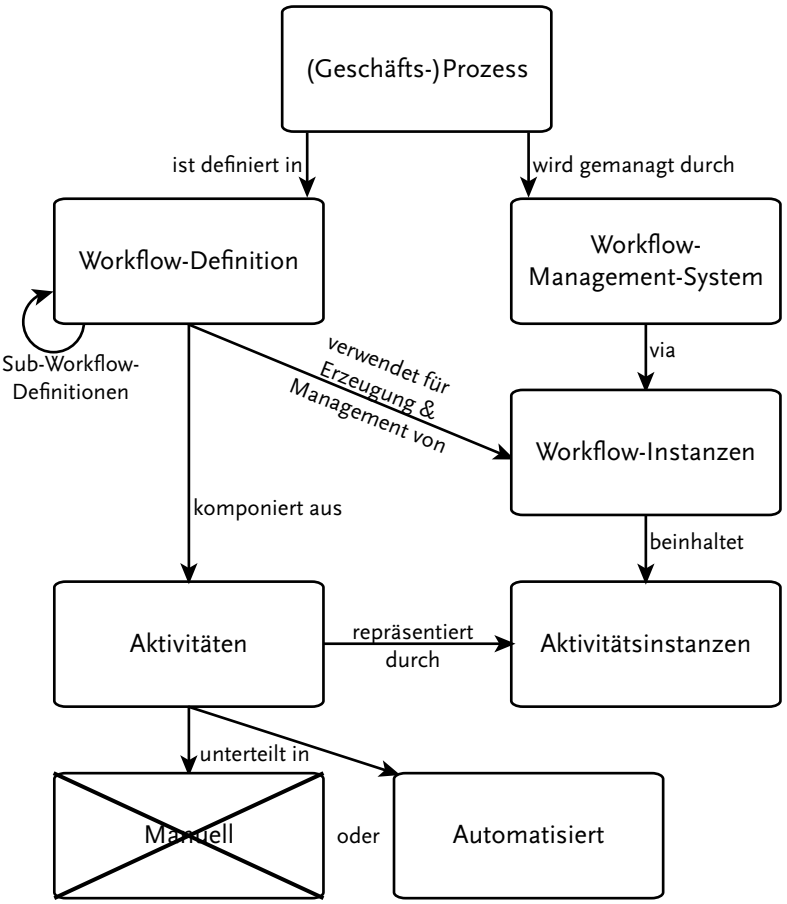


Abbildung 4.1.: Grundlegende Workflow-Terminologie nach Definition der Workflow Management Coalition (basierend auf [WfM99, S. 7])

4.2. Scientific-Workflows

Ein *Scientific-Workflow (S-Wf)* ist eine formale Prozessdefinition zum Erreichen eines wissenschaftlichen Zieles [LAB+09]. S-Wfs zeichnen sich durch spezifische Eigenschaften aus, wie z. B. die Ausrichtung auf Daten und Ressourcen [DGST09]. Wainer bemerkt dazu:

> Whereas office work is about goals, scientific work is about data [WWVM96].

Zudem unterscheidet sich der Lebenszyklus eines Scientific-Workflows von dem eines Business-Workflows.[4] In Abbildung 4.2 auf der folgenden Seite sind beide Lebenszyklen gegenübergestellt, wobei zunächst die ungleiche Anzahl an beteiligten Rollen auffällt (oftmals repräsentiert durch verschiedene Bearbeiter): So sind es beim Business-Workflow typischerweise fünf Rollen, während beim Scientific-Workflow der Wissenschaftler typischerweise allein bzw. im Team mit anderen Wissenschaftlern arbeitet.

[4] Für weitere Unterschiede zwischen Scientific-Workflows und Business-Workflows wird auf [YGN09] verwiesen.

Obwohl sich die durchzuführenden Arbeiten bzw. Phasen nicht wesentlich zu unterscheiden scheinen, ist doch der Umgang mit Workflow-Instanzen sehr verschieden. Insbesondere ist ein häufig anzutreffendes Vorgehen des Wissenschaftlers, dass dieser zur Laufzeit (auf Workflow-Instanz-Ebene) die Ausführung vorzeitig stoppt und eventuell Veränderungen an der Workflow-Definition vornimmt, die sich auf die dann fortzusetzende Instanz auswirken sollen. Entsprechend betrachtet der Wissenschaftler den S-Wf nur als Instanz, bzw. unterscheidet nicht zwischen Workflow-Definition und -Instanz [GSK+11].

Dieses Vorgehen liegt darin begründet, das der S-Wf selbst Gegenstand des wissenschaftlichen Erkenntnisprozesses ist. Ein S-Wf wird oftmals sehr agil variiert in Anzahl, Reihenfolge und Typ von Aktivitäten und Ressourcen. Demgegenüber steht bei Business-Workflows typischerweise eine akribische Modellierungsphasen, so dass die daraus resultierenden Business-Workflow-Instanzen sehr langlebig sind.

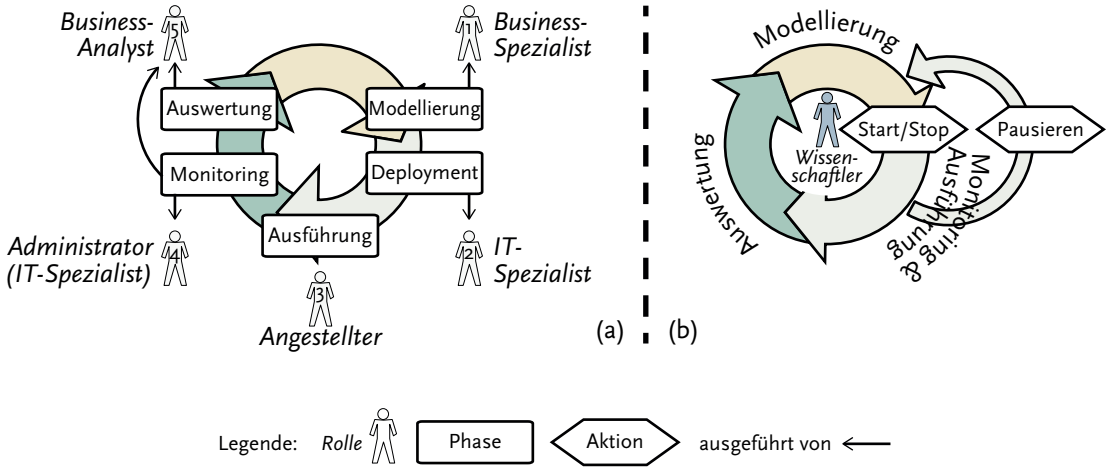


Abbildung 4.2.: Gegenüberstellung der Lebenszyklen eines Business- (a) und Scientific-Workflows (b) unter Beteiligung verschiedener Rollen (basierend auf [GSK+11, S. 5])

4.3. Experimentier-Workflows

In dieser Arbeit wird ein *Experimentier-Workflow (Exp-Wf)* als ein spezifisch strukturierter Scientific-Workflow eingeführt, der die geordneten Phasen *Planung*, *Ausführung* und *Auswertung* umfasst (dargestellt in Abbildung 4.3). Dabei ist eine Besonderheit, dass diese Struktur auch den Lebenszyklus des Exp-Wf darstellt. Man kann sie zudem auch auf das Vorgehen des Experimentators beziehen, um einen Exp-Wf zu erstellen. Der Exp-Wf ist unabhängig von einer konkreten Notationsform. Das primäre Ziel des Exp-Wf ist es, den Prozess des Experimentierens mit computerbasierten Modellen zu automatisieren. Die grundlegende Idee ist daher, sich auf das zentrale Artefakt zu konzentrieren: Das Modell, mit dem experimentiert wird – das MUX. Das MUX muss für die Verwendung in einem Exp-Wf die in Abschnitt 2.2.2 aufgestellten Anforderung erfüllen.

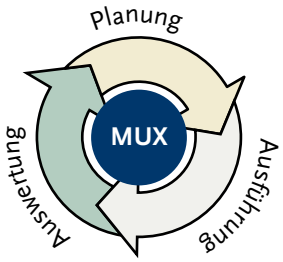


Abbildung 4.3.: Drei Phasen des Experimentier-Workflows

Abbildung 4.4 visualisiert die Komplexität innerhalb von Experimentier-Workflows, die mit verschiedenen Ausführungsumgebungen, Artefakten und Ressourcen umgehen müssen. Typischerweise werden existierende Werkzeuge in einen Exp-Wf eingebunden, die z. B. Daten abfragen, konvertieren, aggregieren oder anderweitig aufbereiten. Die Ausführungsumgebung des MUX wird in einem Exp-Wf geeignet beschrieben, so dass sie für die Ausführung des MUX verwendet werden kann (ansonsten ist ihre Arbeitsweise für den Workflow versteckt). Für solche, verknüpfende Workflows prägte Ludäscher den Begriff *plumbing workflows* [LAB+09]. Dabei müssen insbesondere verschiedene Versionen von Artefakten und Ressourcen berücksichtigt werden und auch der Exp-Wf selbst ist meist im wissenschaftlichen Erkenntnisgewinnungsprozess Veränderungen unterworfen und kann somit in verschiedenen Versionen vorliegen.

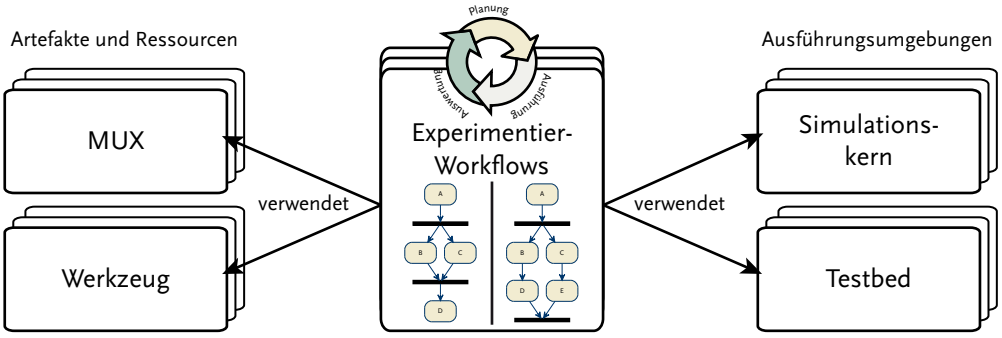


Abbildung 4.4.: Komplexität des Experimentier-Workflows durch Variationen im Ablauf und verschiedener Artefakte und Ressourcen

4.3.1. Planungsphase

Die Aktivitäten in der Planungsphase sind unterteilt in manuelle Aktivitäten zur Entwurfszeit (bevor der Exp-Wf ausgeführt werden kann) und automatisierte Aktivitäten zur Laufzeit (während der Exp-Wf ausgeführt wird).

4.3.1.1. Entwurfszeit

Zur Entwurfszeit des Experimentier-Workflows sind die folgenden manuellen Aktivitäten durchzuführen:

- Das Deklarieren von Ein- und Ausgabeports des MUX.
- Die Definition, wie konkrete Eingabeparameterwerte erzeugt werden. I. A. geschieht dies durch Definition einer systematischen Variation (z. B. einer äquidistanten Schrittweite). Oftmals müssen Eingabeparameterwerte durch Vorverarbeitungen erzeugt werden, z. B. indem Rohdaten aggregiert und/oder von speziellen Ressourcen abgerufen werden.
- Die Definition der zu beobachtenden Ausgaben.
- Wie und in welcher Ausführungsumgebung das MUX-Programm auszuführen ist.
- Die Aktivitäten zur Auswertung der beobachteten Ausgaben.

4.3.1.2. Laufzeit

Zur Laufzeit kann es zu einer Planungsphase kommen, die automatisch durch das WfMS ausgeführt wird. Das ist beispielsweise der Fall, wenn Feedback zwischen Experimenten ausgewertet wird, d. h., auf der Basis bisher erlangter Experimentergebnisse eine Menge von weiteren Experimenten geplant wird.

4.3.2. Ausführungsphase

Auf Basis der Exp-Wf-Definition läuft die Ausführungsphase innerhalb des Workflow-Management-Systems vollautomatisch ab. Neben der Ausführung des MUX-Programmes in seiner spezifizierten Ausführungsumgebung werden die folgenden, automatischen Aktivitäten veranlasst:

1. Transformation der Eingabeparameterwerte aus der Exp-Wf-Definition in ein Format, das zur Übergabe an die Ausführungsumgebung geeignet ist (z. B. Kommandozeilenparameter oder Key-Value-Textdateien),
2. Ausführen des MUX-Programmes in der spezifizierten Ausführungsumgebung (z. B. ein Simulationskern) und
3. Einlesen der Ausgabewerte des MUX-Programmes aus dem spezifischen Ausgabeformat der Ausführungsumgebung und Speicherung in der Exp-Wf-Instanz.

Insbesondere Punkt zwei kann multiple, vorangestellte Aktivitäten bedingen. So muss das MUX möglicherweise erst zu einem MUX-Programm transformiert werden, woraus ein ausführbares Programm erzeugt wird (i. A. als Simulationsprogramm im Simulator).

4.3.3. Auswertungsphase

Durch die Aktivitäten in der Auswertungsphase werden die Modellbeobachtungen einer MUX-Programmausführung verarbeitet. Aufgrund der Arten der Aktivitäten in der Auswertungsphase kann man zwischen manueller, semi- und vollautomatischer Auswertungsphase unterscheiden.

4.3.3.1. Manuelle Auswertungsphase

Hierbei finden nur manuelle Aktivitäten statt, die nicht im Exp-Wf definiert sind. Eine manuelle Auswertungsphase ist z. B. was der Experimentator oft lapidar mit „drauf schauen" bezeichnet. Dabei inspiziert er die Resultate der Experimente und sucht nach Dingen, die ihm auffällig erscheinen – oftmals sind solche Auffälligkeiten widersprüchlich zu impliziten Erwartungen. Ein solches Vorgehen findet man häufig in frühen Stadien des Experimentierens, in denen die Ziele bzw. zu beantwortenden Fragestellungen noch nicht klar sind.

4.3.3.2. Semi-automatische Auswertungsphase

Eine semi-automatische Auswertungsphase ist die am häufigsten anzutreffende Form. Eine typische Aktivität hierbei ist z. B. das Berechnen einer Aggregation, die dann in einem Diagramm dargestellt werden könnte. Damit werden typischerweise auftretende, große Mengen von Experiment-Resultaten handhabbar, die andernfalls nicht mehr mittels einer manuellen Auswertungsphase bearbeitet werden können.

4.3.3.3. Vollautomatische Auswertungsphase

Die Bezeichnung „vollautomatisch" mag zunächst irreführend wirken, wenn man ihn auf den wissenschaftlichen Erkenntnisprozess beziehen will. Er ist jedoch auf eine Menge von Experimenten bezogen, wobei unter Verwendung definierter Aktivitäten und unter Einbeziehung des Untersuchungszieles Modellbeobachtungen ausgewertet werden. Dies kann innerhalb des M&S-Zyklusses wiederum zu neuen Experimenten führen (oder zur Beendigung, wenn das Untersuchungsziel erreicht wurde). Solche Aktivitäten sind typischerweise die Berechnung einer spezifischen Metrik bzw. die Auswertung einer *Fitness-Funktion* (oder auch *Zielfunktion*). Eine Fitness-Funktion f bildet dabei eindeutig eine Menge von Beobachtungen O eines Experiments auf einen skalaren Wert s ab, so dass ein qualitativer Vergleich von Experimenten untereinander möglich wird: $f(O) = s$ Ein solcher Vergleich kann dann durch eine weitere Aktivität im Exp-Wf definiert sein, die z. B. ein Diagramm bzw. Textuellen Report erstellt. Im Kontext einer vollautomatischen Auswertung finden Fitness-Funktionen Anwendung innerhalb genetischer Algorithmen [SHF94]. Dabei werden neue Experimente geplant, indem jeweils eine neue Menge von Eingabeparameterwerten aus dem Ergebnis der Fitness-Funktion eines vorherigen Experiments berechnet wird.

5. Workflow-Systeme

Dieses Kapitel stellt die Bestandteile eines Workflow-Systems vor. Sie sind für das spätere Verständnis des in dieser Arbeit vorgestellten neuen Ansatzes eines *Experimentier*-Workflow-Systems notwendig. Nachdem Kapitel 4 die Definition und Arten von Workflows zum Inhalt hatte, beschäftigt sich dieses Kapitel mit deren Verwaltung durch Workflow-Management-Systeme. Der Begriff *Workflow-System* bezeichnet die Gesamtheit von Workflow-Definitionen, zugehörigen -Instanzen und dem Workflow-Management-System, das sie verwaltet.[1]

5.1. Das Workflow-Referenzmodell der Workflow Management Coalition

Das Workflow-Referenzmodell der WfMC [WfM99, S. 23] ist ein Architekturmodell des Workflow-Management-Systems und beschreibt dessen Schnittstellen und Bestandteile (dargestellt in Abbildung 5.1 auf der nächsten Seite). Hierbei umfassen die Schnittstellen eines WfMS zu seiner Umgebung folgende fünf Bereiche:

- den Im-/ und Export von Prozessdefinitionen,
- die Interaktion mit Client-Applikationen (siehe Abschnitt 5.1.4),
- das Aufrufen von Software-Werkzeugen oder Applikationen,
- Interoperabilität zwischen verschiedenen Workflow-Management-Systemen und
- Administration und Monitoring-Funktionalität.

5.1.1. Workflow-Management-System

Ein *Workflow-Management-System (WfMS)*[2] ermöglicht die Definition, die Erzeugung und die Ausführung von Workflows durch den Einsatz von Software, die auf einer oder mehrerer *Workflow-Engines* basiert und die in der Lage sind, […] eventuell benötigte, weitere Software-Werkzeuge und Applikation aufzurufen (übersetzt aus [WfM99, S. 9]).

---

[1] Diese Unterscheidung ist somit analog zur Differenzierung von Datenbanksystem und Datenbankmanagementsystem. Doch im Gegensatz zur Datenbank-Community, wird diese Differenzierung in der Workflow-Literatur oftmals nicht vorgenommen.

[2] Kreplin übersetzt diesen Begriff ins Deutsche mit *Vorgangssteuerungssystem* [Kre99].

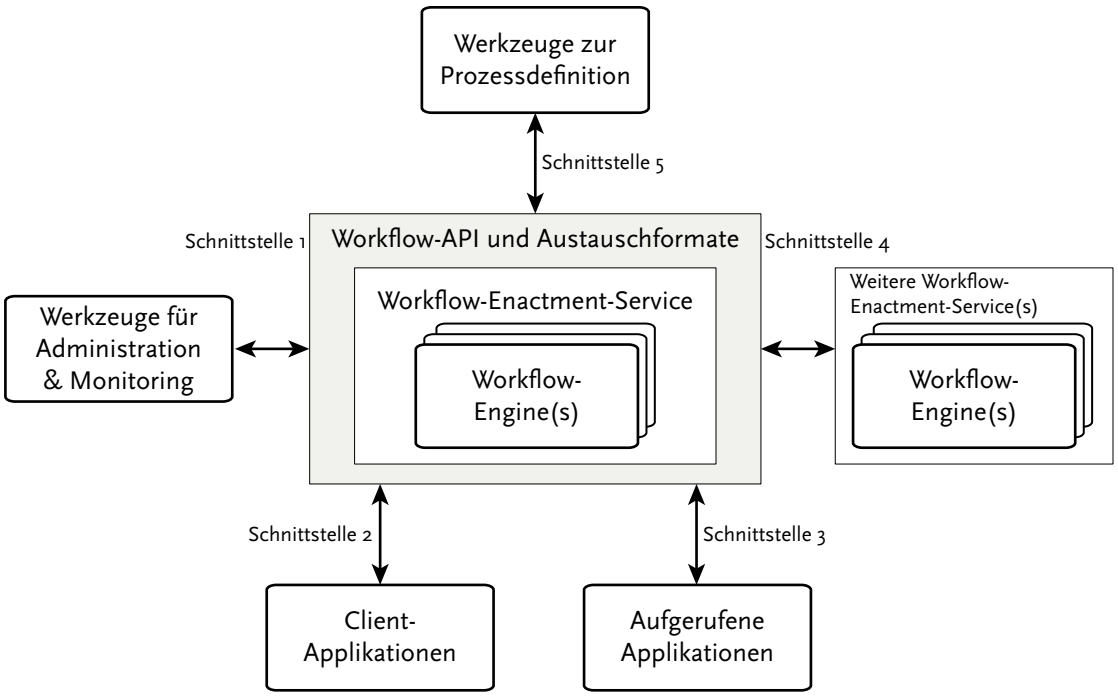


Abbildung 5.1.: Bestandteile eines Workflow-Management-Systems: Das Workflow-Referenzmodell der WfMC (basierend auf [WfM99, S. 23])

5.1.2. Workflow-Enactment-Service

Der Begriff *Workflow-Enactment* bezeichnet die Prozessinstanziierung. Der *Workflow-Enactment-Service*[3] ist entsprechend ein Software-Dienst, der aus einer oder mehrerer *Workflow-Engines* (siehe Abschnitt 5.1.3) besteht und einzelne Workflow-Instanzen erzeugt, managt und ausführt (übersetzt aus [WfM99, S. 59]).

5.1.3. Workflow-Engine

Eine *Workflow-Engine (Wfe)*[4] ist ein Software-Dienst, der die Laufzeit-Ausführungsumgebung für eine Prozessinstanz bereitstellt (übersetzt aus [WfM99, S. 57]). Nach Meinung des Autors der vorliegenden Arbeit ist es fraglich, warum hier von einer Prozessinstanz anstelle einer Workflow-Instanz gesprochen wird. Da ein Workflow-Enactment-Service aus einer oder mehrerer Workflow-Engines zusammengesetzt ist (siehe Abschnitt 5.1.2) und dessen Definition die Behandlung von Workflow-Instanzen beinhaltet, sollte folglich auch bei Workflow-Engines von der Behandlung von Workflow-Instanzen die Rede sein.

[3] Kreplin übersetzt diesen Begriff ins Deutsche mit *Prozessbearbeitungsservice* [Kre99].

[4] Kreplin übersetzt diesen Begriff ins Deutsche mit *Vorgangssteuerungssystem* und trifft somit keine sprachliche Unterscheidung zur Übersetzung von *Workflow-Management-System* [Kre99].

5.1.4. Workflow-Applikation

Der Begriff *Workflow-Applikation* bezeichnet ein Computersystem, das mit einem Workflow-Enactment-Service interagiert, um Teile der Ausführung einer (oder mehrerer) Aktivität(en) durchzuführen (übersetzt aus [WfM99, S. 41]). Im Workflow-Referenzmodell wird zwischen zwei Kategorien von Workflow-Applikationen unterschieden: *Client-Applikationen* und *Aufgerufene Applikationen*. Client-Applikationen benutzen von einer Workflow-Engine angebotene Funktionalität. Demgegenüber erbringen Aufgerufene Applikationen Funktionalität für das Workflow-Management-System.

5.2. Klassen von Workflow-Daten

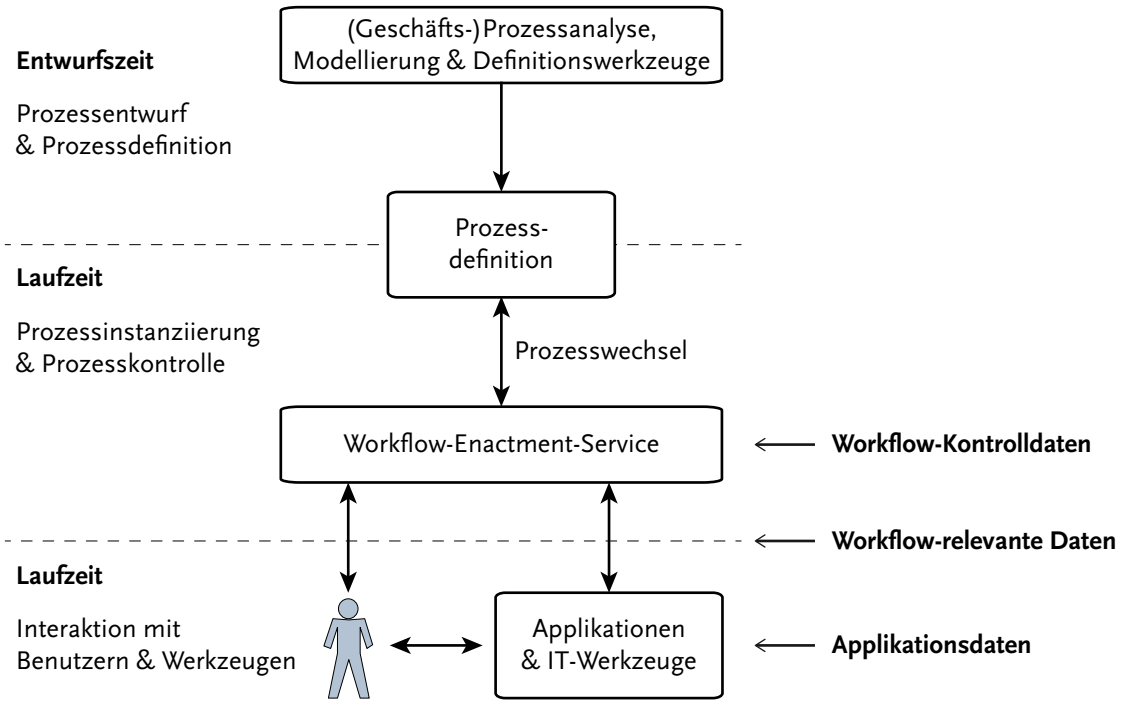


Abbildung 5.2.: Klassen von Workflow-Daten (basierend auf [WfM99, S. 44])

5.2.1. Workflow-Kontrolldaten

Workflow-Kontrolldaten[5] repräsentieren den dynamischen Zustand des Workflow-Systems und seiner Prozessinstanzen[6] (übersetzt aus [WfM99, S. 46]). Workflow-Kontrolldaten werden

[5] Kreplin übersetzt diesen Begriff ins Deutsche mit *Vorgangssteuerungsdaten* [Kre99].

[6] Auch hierbei müsste man genauer von Workflow-Instanzen sprechen.

vom Workflow-Management-System und/oder einer Workflow-Engine gemanagt. Sie sind nur intern dem WfMS zugänglich und normalerweise nicht von Applikationen zugreifbar. Beispiele für Workflow-Kontrolldaten sind:

- Zustandsinformationen über jede Workflow-Instanz,
- Zustandsinformationen über jede Aktivitätsinstanz (aktiv oder inaktiv) oder
- Informationen über Fehlererholung und Fortsetzungspunkte innerhalb eines Prozesses.

5.2.2. Workflow-relevante Daten

In [WfM99, S. 45] sind *Workflow-relevante Daten*[7] als Daten definiert, die den Zustandsübergang einer Workflow-Instanz charakterisieren [WfM99, S. 49]. Der Zustand einer Workflow-Instanz ist bestimmt durch die Zustände der zugehörigen Aktivitätsinstanzen. Der Zustandsübergang (*state transition*) einer Workflow-Instanz ist hierbei der Übergang von einem internen Zustand zu einem anderen, wobei dadurch eine Zustandsänderung bewirkt wird, wie z. B. das Initiieren einer bestimmten Aktivität [WfM99, S. 49]. Eine zeitliche Abfolge von solchen Zustandsänderungen repräsentiert den Fortschritt der Ausführung der Workflow-Instanz und kann als Monitoring-Information (*audit data*) von der Workflow-Engine erfasst und ausgewertet werden.

Workflow-relevante Daten können sowohl durch Workflow-Applikationen als auch durch Workflow-Engines manipuliert werden. Insbesondere können sie auch innerhalb von Workflow-Aktivitätsinstanzen aus zugegriffen werden. Diese Aktivitätsinstanzen müssen nicht notwendigerweise zur selben Workflow-Instanz gehören. Dadurch kann eine Aktivitätsinstanz ihr Verhalten entsprechend des bisherigen Workflow-Fortschrittes anpassen. Workflow-relevante Daten werden unterschieden in getypte und ungetypte Daten. Getypte Daten sind dem WfMS in ihrer Struktur bekannt und können von ihm verarbeitet werden. Demgegenüber sind ungetypte Daten lediglich durch das WfMS referenzierbar und können an Workflow-Applikationen weitergeleitet werden.

5.2.3. Applikationsdaten

Applikationsdaten sind anwendungsspezifisch und nicht von einem Workflow-Management-System zugreifbar (übersetzt aus [WfM99, S. 44]). Das WfMS besitzt keinerlei Kenntnis über die Existenz von Applikationsdaten.

5.3. Klassifikation

Viele Artikel in der wissenschaftlichen Literatur bezüglich der Klassifikation von Workflow-Systemen vergleichen innerhalb einer bestimmten Subklasse, wie z. B. der von Geschäftspro-

[7] Kreplin übersetzt diesen Begriff ins Deutsche mit *Vorgangsdaten* [Kre99].

zessen oder von wissenschaftlichen Prozessen. Anhand der Subklasse wird häufig auf sich daraus ergebende Anforderungen weiter fokussiert, wie z. B. auf Grid-Computing bei wissenschaftlichen Prozessen [YB05].

Ziel dieses Abschnittes ist es, zunächst eine Klassifikation von Workflow-Systemen zu geben, die unabhängig von einer bestimmten Subklasse (und Fokussierung) ist. Hierauf aufbauend zeigt Abschnitt 5.5, wie Scientific-Workflow-Systeme gegenwärtig für Design und Management von Experimentier-Workflows eingesetzt werden, und welche Probleme sich daraus ergeben. Dieser Ansatz wird in Abschnitt 7.5.3 mit Simulationssystemen hinsichtlich ihrer Eignung für das Design und Management von Experimentier-Workflows verglichen.

5.3.1. Workflow-Definitionsmodell

Ein Workflow kann als gerichteter Graph W ohne Mehrfachkanten aufgefasst werden, der aus Aktivitäten (Knoten V) und ihren Transitionen (gerichtete Kanten E) besteht. In [LAB+09, S. 4] gibt Ludäscher die folgende, mathematische Notation dieses Aktivitäten-Transitionen-Graphen W:

> Mit W verknüpft ist eine Menge von Parametern $\overline{p}$, Eingabedatensätzen $\bar{x}$ und Ausgabedatensätzen $\overline{y}$. Ein Workflow-Definitionsmodell M beschreibt, wie die parametrisierte Workflow-Instanz $W_{\overline{p}}$ unter Verwendung von $\bar{x}$ die Ausgabe $\overline{y}$ erzeugt. Formal aufgeschrieben definiert M folgende Abbildung: $M : \mathcal{W} \times \bar{P} \times \bar{X} \rightarrow \bar{Y}$ und produziert für jeden Workflow $W \in \mathcal{W}$, Parametersatz $\bar{p} \in \bar{P}$, Eingabedatensatz $\bar{x} \in \bar{X}$ eindeutig die Ausgaben $\bar{y} \in \bar{Y}$. Die Kurzschreibweise hierfür sei $\bar{y} = M(W_{\overline{p}}(\bar{x}))$.

Die gerichteten Kanten des Graphen W repräsentieren Transitionen zwischen Aktivitäten. Die Transitionsrichtung definiert dabei die Reihenfolge, in der Aktivitäten des Workflows ausgeführt werden. Dadurch wird der Kontrollfluss innerhalb des Workflows festgelegt. Eine stärkere Semantik der gerichteten Kanten ist, dass durch sie der Datenfluss zwischen Aktivitäten bestimmt wird. Dabei implizieren Datenflussabhängigkeiten i. A. auch Kontrollflussabhängigkeiten.

Auf Basis dieser Graph-Repräsentation kann man zwei Klassen von Workflow-Definitionsmodellen unterscheiden: werden keine Zyklen unterstützt, basieren sie auf einem *Directed-Acyclic-Graph (DAG)*. Werden hingegen Zyklen unterstützt, spricht man von einem *Directed-Cyclic-Graph (DCG)*. Die Mehrheit der Workflow-Systeme ist DAG-basiert und stellt spezielle Container-Aktivitäten bereit, die Schleifen über darin enthaltene Sub-Aktivitäten erlauben. Dadurch entstehen in der Praxis kaum Einschränkung in der Ausdrucksmächtigkeit gegenüber DCG-basierten Systemen.

5.3.2. Perspektiven eines Workflow-Definitionsmodells

Zur Erleichterung des Umganges mit potentiell sehr komplexen Workflow-Definitionsmodellen ist es unter Umständen sinnvoll, nur funktionale Teilbereiche betrachten zu müssen.

In der Praxis haben sich dabei vier Teilbereiche bzw. Perspektiven als besonders relevant herausgestellt. Leymann et al. [LR00] sprechen hierzu von den *Three Dimensions of Workflow – (What, Who, Which)*. In [ATHKB03] wird dies zu vier Perspektiven von Workflows erweitert: Kontrollfluss-Perspektive, Daten-Perspektive, Ressourcen-Perspektive und operative Perspektive.

Kontrollfluss-Perspektive

Die statische Kontrollfluss-Perspektive betrachtet den Aktivitäten-Transitionen-Graphen des Workflow-Definitionsmodells (siehe Abschnitt 5.3.1). Dabei werden dynamische Änderungen des Kontrollflusses zur Laufzeit nicht berücksichtigt, die beispielsweise durch die Behandlung von aufgetretenen Ausnahmen verursacht werden. Ein wichtiger Aspekt der statischen Kontrollfluss-Perspektive ist die Ermöglichung maximaler Parallelisierbarkeit von Aktivitäten (unter Beibehaltung der semantischen Korrektheit des Prozesses).

Im Gegensatz zur Ressourcen-Perspektive ist die Modellierung der Kontrollfluss-Perspektive von Workflow-Modellen gut erforscht. Wichtige Beiträge hierzu finden sich in den Ergebnissen einer Arbeitsgruppe um van der Aalst, die durch Erarbeitung eines Katalogs von statischen Kontrollfluss-Mustern (*workflow patterns*) eine Definition aktueller Anforderungen an die Kontrollfluss-Perspektive von Workflows bereitstellt [ATHKB03].

Daten-Perspektive

Bei der Daten-Perspektive handelt es sich insofern um eine Erweiterung der Kontrollfluss-Perspektive, weil Daten entlang des Aktivitäten-Transitionen-Graphen betrachtet werden. Diese Daten sind Workflow-relevante Daten in den Bedingungen von Transitionen sowie Anwendungsdaten, die entlang des Aktivitäten-Transitionen-Graphen fliessen. Dieser Erweiterungscharakter ist eventuell der Grund, dass die Daten-Perspektive von Leymann et al. [LR00] nicht als eigenständige Perspektive benannt wird.

Ressourcen-Perspektive

Die Ressourcen-Perspektive beschäftigt sich mit der Zuordnung von Bearbeitern zu Aktivitäten. Ein häufig angewendetes Vorgehen dabei ist, zunächst eine abstrakte Zuordnung gemäß erforderlicher Qualifikationen und Berechtigungen zu modellieren, wobei Bearbeiter abstrakt als Ressourcen betrachtet werden. Darauf aufbauend werden Regeln formuliert, mit Hilfe derer die Abbildung zwischen abstrakten Ressourcen und realen Mitarbeitern erfolgt. Die Durchführung dieser zweiten Abbildung wird häufig als *staff resolution* bezeichnet.

Operative Perspektive

Bestandteil der operativen Perspektive ist die Bindung zwischen (automatisierten) Aktivitäten und Anwendungen. Aspekte dieser Bindung sind zum Beispiel technische Details des Anwendungsaufrufes sowie der Datenübergabe.

5.3.3. Workflow-Kontrollflussmuster

In [ATHKB03] werden 20 Workflow-Kontrollflussmuster (*workflow patterns*) identifiziert, aufgeteilt in 10 Basismuster und 10 komplexe Muster. Dabei werden die 10 Basismuster von fast allen untersuchten Workflow-Management-Systemen unterstützt, einige der komplexeren Muster jedoch von keinem einzigen. Komplexe Muster, die auf der gleichzeitigen Ausführung mehrerer Aktivitätsinstanzen des selben Typs beruhen, bereiten den in der Studie untersuchten WfMS große Probleme, insbesondere, wenn die Anzahl der Aktivitätsinstanzen erst zur Laufzeit bekannt ist.

Van der Aalst betrachtet nur statischen Kontrollfluss, so dass keine Muster für Ressourcen-Allokation, *case handling*, Ausnahmebehandlung und Transaktionsmanagement berücksichtigt werden. Die Basismuster orientieren sich an den elementaren Kontrollflusskonzepten der Workflow Management Coalition, zu denen zählen [WfM99, S. 29–36]:

sequentieller Ablauf Ein Segment einer Workflow-Instanz, deren Aktivitätsinstanzen hintereinander und innerhalb einer einzigen Kontrollinstanz ablaufen (der eines WfMS). In [ATHKB03] wird dies als Sequenz bezeichnet (Muster 1).

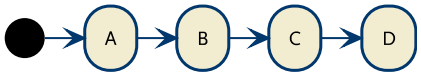
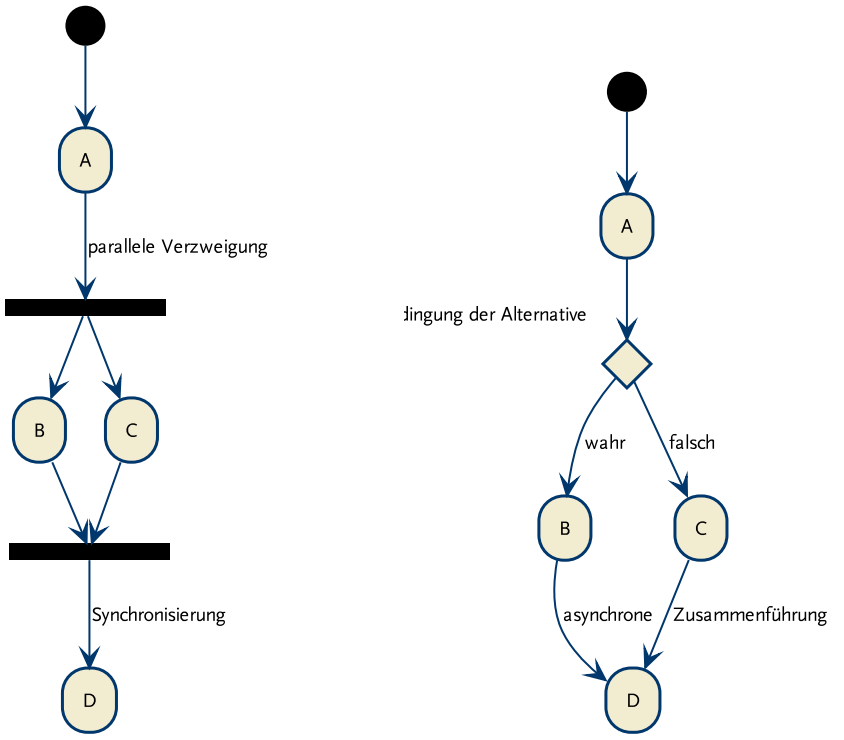


Abbildung 5.3.: Workflow-Kontrollflussmuster: Sequentieller Ablauf (als UML-Aktivitätsdiagramm)

paralleler Ablauf Ein Segment einer Workflow-Instanz, in der zwei oder mehr Aktivitätsinstanzen parallel innerhalb mehrerer Kontrollinstanzen eines Workflow-Management-Systems ablaufen. Ein paralleler Ablauf beginnt mit einer parallelen Verzweigung (*AND-Split*, Muster 2 in [ATHKB03]) und endet mit einer Synchronisierung (*AND-Join*, Muster 3 in [ATHKB03]).

Alternative (*OR-Split*) Ein Punkt innerhalb des Workflows, zu dem der Kontrollfluss mehrere, alternative Pfade nehmen kann. Der eingeschlagene Pfad zu einer nachfolgenden Aktivität (B oder C in Abbildung 5.4b) wird anhand der Auswertung von Bedingungen (über Workflow-Kontrolldaten) ausgewählt, die an den Zustandsübergängen der Ausgangsaktivität (A in Abbildung 5.4b) annotiert sind. In [ATHKB03] wird dies bezeichnet als Muster 4, Exklusive Auswahl (*exclusive choice*) und unterscheidet dazu Muster 6, Mehrfachauswahl (*multi-choice*), bei der eine Menge von Zweigen zu mehreren nachfolgenden Aktivitäten ausgewählt werden können.

asynchrone Zusammenführung (*OR-Join*) Ein Punkt innerhalb des Workflows, an dem zwei oder mehrere, alternative Aktivitätszweige wieder zusammengeführt werden, so dass eine einzelne Aktivität (D in Abbildung 5.4b) im nächsten Schritt der Workflow-Ausführung abgearbeitet wird.



(a) Paralleler Ablauf

(b) Alternative mit asynchroner Zusammenführung

Abbildung 5.4.: Workflow-Kontrollflussmuster: Paralleler Ablauf und Alternative (als UML-Aktivitätsdiagramme)

Iteration Die zyklisch wiederholte Ausführung einer oder mehrerer Aktivitäten (C in Abbildung 5.4b), bis eine bestimmte Bedingung erfüllt ist (siehe nachfolgend: Vor- und Nachbedingung).

Vorbedingung Ein logischer Ausdruck innerhalb der Workflow-Definition, der von der Workflow-Engine ausgewertet wird und darüber entscheidet, ob nachfolgende Aktivitäten gestartet werden.

Nachbedingung Ein logischer Ausdruck innerhalb der Workflow-Definition, der von der Workflow-Engine ausgewertet wird und darüber entscheidet, ob die zugehörige Aktivitätsinstanz vollständig abgearbeitet wurde.

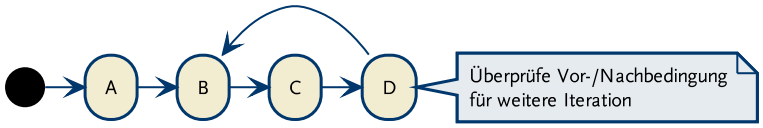


Abbildung 5.5.: Workflow-Kontrollflussmuster: Iteration (als UML-Aktivitätsdiagramm)

5.4. Relation zur Terminologie von Modellierung & Simulation

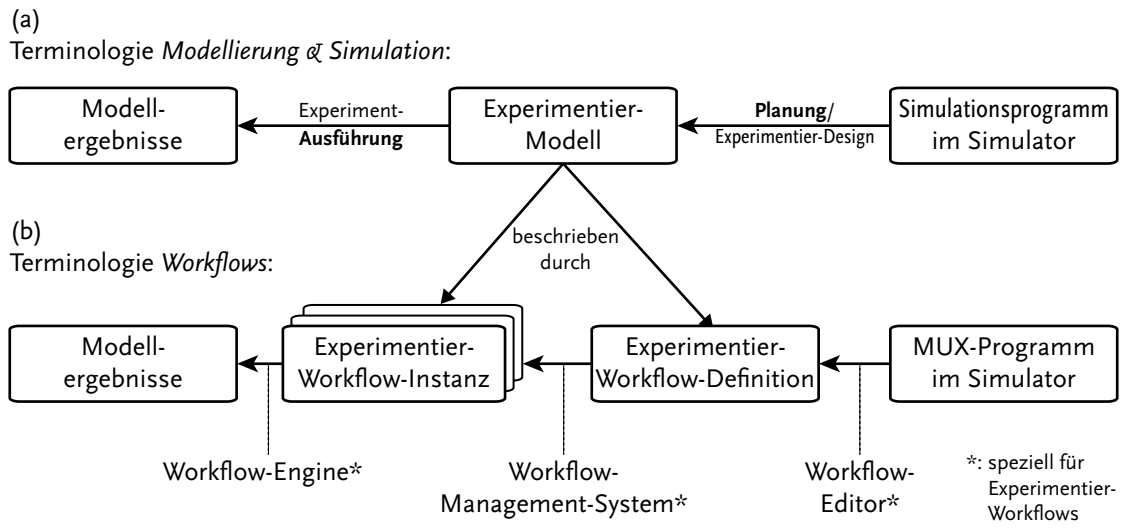


Abbildung 5.6.: Workflow-Terminologie zur Beschreibung von Experimentier-Prozessen (Teilabbildung (a) entstammt Abbildung 2.3 auf Seite 25)

Nachdem sowohl die Terminologien von Modellierung & Simulation (Kapitel 2) als auch von Workflows (Kapitel 4 und 5) eingeführt wurden, wird in diesem Abschnitt die Verbindung beider Terminologien erläutert.

Ausgangspunkt ist das MUX-Programm bzw. Simulationsprogramm im Simulator, das in der rechten Seite von Abbildung 5.6 dargestellt ist. Mit Hilfe eines speziellen Workflow-Editors wird eine Experimentier-Workflow-Definition erstellt, welche die Ausführung des Simulators und potentiell weitere Aktivitäten spezifiziert. Die Experimentier-Workflow-Definition beschreibt generell Aktivitäten, Kontrollflüsse zwischen diesen Aktivitäten und Ressourcen, die zur Beschreibung eines Experimentier-Prozesses benötigt werden. Sie ist ein Teil des auf Seite 28 eingeführten Experimentier-Modells. Der zweite Teil des Experimentier-Modells besteht aus den Experimentier-Workflow-Instanzen, die auf der Basis der Experimentier-Workflow-Definition durch ein spezielles Workflow-Management-System erzeugt werden. Experimentier-Workflow-Instanzen werden durch eine spezielle Workflow-Engine ausgeführt.

5.5. Scientific-Workflow-Systeme für Experimentier-Prozesse

Es existieren viele Scientific-Workflow-Systeme, die jeweils spezifische Aktivitäten anbieten, um sich für wissenschaftliche Prozesse in einer bestimmten Anwendungsdomäne zu empfehlen. Beispiele für solche Systeme sind:

- Taverna [OAF+04], das vor allem in der Biologie, Chemie und Bioinformatik eingesetzt wird,

- Kepler [LAB+06], dessen hauptsächliche Anwendungsbereiche in der Ökologie und Geologie liegen,
- Pegasus [DSS+05] wird in der Astronomie und Physik eingesetzt und
- Askalon [FPD+07] ist ein Grid- und Cloud-Computing S-Wf-System, mit dem z. B. Workflows in der Hydrologie unterstützt werden.

S-Wf-Systeme sind zwar flexibel, jedoch oftmals zu komplex in der Handhabung, um von Laien benutzt zu werden, die mit Workflow-Konzepten nicht vertraut sind. Das *California Earthquake Center* hat deshalb ein eigenes Computersystem entworfen, das von existierenden S-Wf-Systemen (u. A. Pegasus) abstrahiert und Konzepte bereitstellt, die näher an den Bedürfnissen von Seismologen orientiert sind [MCD+05]. Mit diesem System werden beispielsweise Erdbebenbedrohungskarten[8] erstellt, indem Schablonen von Workflow-Definitionen für deren Berechnung zur Verfügung gestellt werden.

Ähnlich verhält es sich mit dem Experimentieren auf der Basis von Modellen, das sich als verändernder, iterativer Prozess mit variabel komponierbaren Teilaufgaben darstellt. Solche Prozesse können mit S-Wf-Systemen aufgrund ihrer Flexibilität gehandhabt werden. Allerdings werden dabei die speziellen Eigenschaften und Möglichkeiten beim Experimentieren mit Modellen nur unzureichend berücksichtigt (z. B. können u. A. Variationen von Eingabeparameterbelegungen nicht adäquat ausgedrückt werden; siehe Abschnitt 1.2). Ihre Handhabung ist zudem komplex und erfordert eine den Experimentatoren oftmals fremde Denkweise, bei der sie ihre Experimente mit Daten- und Kontrollflüssen ausdrücken müssen, um S-Wf-Systeme nutzen zu können.

Ein einfaches und praxisnahes Beispiel soll dies verdeutlichen. Dabei soll ein Modell eines neuen Routing-Protokolls – das MUX in diesem Beispiel – in einem simulierten Computernetzwerk untersucht werden (dieses Szenario ist Teil des NetTopo-Beispiels im Anhang, Abschnitt B.3 auf Seite 178). Das MUX hat drei Eingabeparameter: eine Netzwerktopologie (mit räumlich positionierten Netzwerkknoten und zwischen ihnen mögliche Übertragungsverbindungen), ein Parameter zur Auswahl eines im MUX enthaltenen Radiomodells und eine Bitrate, mit der eine bestimmte, vordefinierte Übertragung durchgeführt werden soll. Die Modellausgaben sind Zeitreihen von Modellzeit und dabei jeweils erfolgtem Durchsatz der Übertragung. Es wird angenommen, dass aus dem MUX ein Simulator erzeugt wurde.

Das Experimentieren in diesem Beispiel besteht in der Variation der Belegungen für die drei Eingabeparameter des MUX, die zur Vereinfachung aus vorgegebenen Mengen erfolgen soll. In Taverna kann man das Beispiel wie in Abbildung 5.7 als Workflow modellieren, der Parameter für die Ein- und Ausgabe besitzt, die analog zu denen des MUX sind. Es gibt zwei Aktivitäten: eine bildet aus den Eingabeparametermengen das Kreuzprodukt (CartesianProdukt) und die andere führt den Simulator aus (Execute_Simulator), der jeweils ein Element der Kreuzproduktmenge übergeben bekommt.

[8] Erdbebenbedrohungskarten (*probabilistic seismic hazard analysis map*) zeigen für eine bestimmte Region der Erde die Wahrscheinlichkeit, dass in den nächsten (typischerweise) 50 Jahren eine bestimmte Erdbodenbeschleunigung erreicht wird.

Das Beispiel soll zeigen, dass man durch diese Art der Beschreibung nur eine minimale strukturelle Ordnung im Ablauf erreicht hat. Die Semantik des Experiments ist in den Aktivitätsdefinitionen versteckt, die in Java zu spezifizieren sind. Das ist vergleichbar mit Methodenaufrufen, die Parameterwerte übergeben bekommen. Artefakte, wie das MUX, das MUX-Programm und die Ausführungsumgebung, werden nicht explizit beschrieben. Woher die Artefakte stammen, in welcher Version sie vorliegen und ob sie persistent referenzierbar sind, kann der Beschreibung in Taverna nicht entnommen werden, ohne in den Java-Quellcode der Aktivitäten Einblick zu nehmen.

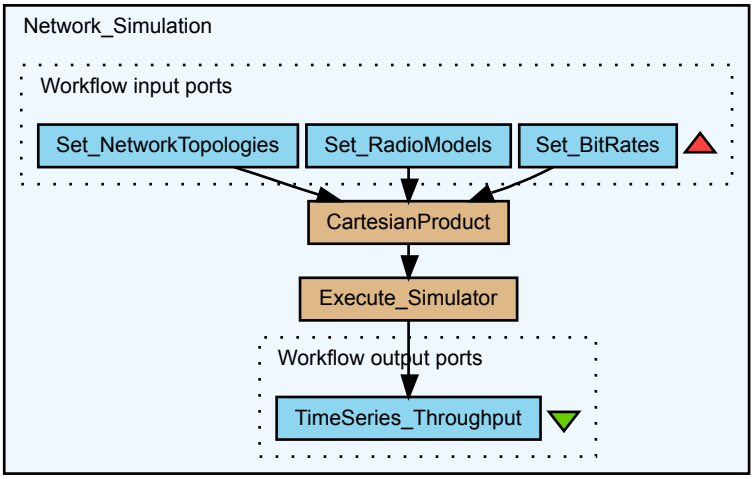


Abbildung 5.7.: Beispiel eines einfachen Experimentier-Workflows in der grafischen Darstellung von Taverna, das einen Teil des NetTopo-Beispiels zeigt (siehe Abschnitt B.3 auf Seite 178). Aktivitäten sind in Form von bräunlichen Kästen dargestellt und Parameter in Form von blauen.

Dem umgekehrten Weg einer Verbindung von Workflows und Simulationen wurde in der Diplomarbeit von Sauer nachgegangen [Sau06]. In dieser Arbeit wurde untersucht, inwiefern Geschäftsprozesse, die in Form von standardbasierten Workflow-Modellen vorliegen, mittels der Simulationsbibliothek ODEMx ausführbar gemacht werden können (beispielhaft für BPMN). Auf der Basis dieser Vorarbeiten wäre vielleicht auch ein Ansatz möglich, mit dem ODEMx um Scientific-Workflow-Konzepte erweitert werden könnte, um eine Experimentierunterstützung anzubieten.

6. Domänenspezifische Sprachen

Sprache an sich hat den generellen Zweck, Informationen zu transportieren. Vom Umgang mit Menschen kennt man, dass verschiedene Gruppen von Personen eine Sprache mit ausgewähltem Vokabular in ihren Konversationen verwenden, um den Personen einer bestimmten Gruppe Informationen mitzuteilen. Solche Gruppen können z. B. durch eine permanente Zugehörigkeit zu einer Berufsgruppe (z. B. Informatiker) oder durch eine temporäre Zugehörigkeit (z. B. Besuch eines Konzertes) charakterisiert sein. Es ergibt sich somit ein sozialer Kontext, in dem die Konversation abläuft. Selbstverständlich hat die Konversation auch ein Thema zum Inhalt, das wiederum bestimmte Fachtermini beinhalten kann. Beides zusammen, Fachtermini und das gemeinsame Vorwissen durch den persönlichen Hintergrund der beteiligten Personen eines sozialen Kontextes, kann man intuitiv als *Domäne* des Gespräches verstehen. Man könnte folglich die Sprache, die in einem solchen Gespräch verwendet wird, als *domänenspezifische Sprache* bezeichnen. In diesem Zusammenhang steht der Begriff *Domäne* für *Wissensdomäne*. Eine Wissensdomäne ist ein Wissensgebiet, das definiert ist durch die Gesamtheit des Wissens innerhalb eines Fachbereichs. Ein solcher Fachbereich zeichnet sich oft durch eine Fachsprache aus, und es existieren ausgewiesene Experten in diesem Fachbereich. Diese Vorüberlegungen sollen als Einleitung für dieses Kapitel 6 dienen, das (wie auch die gesamte Arbeit) nicht von natürlichen Sprachen, sondern von *Computersprachen* handelt. Entsprechend wird im Weiteren *Sprache* als Synonym für *Computersprache* verwendet.

6.1. Motivation

Computersprachen sind alle Sprachen, mit deren Hilfe Informationen an einen Computer oder zwischen Computern übermittelt werden. Die Subklasse von Computersprachen, die für einen bestimmten, spezifischen Anwendungsbereich – eine *Domäne* – entwickelt sind, werden als *Domain-Specific-Languages (DSLs)* bezeichnet [FP10].[1] Demgegenüber stehen sogenannte *General-Purpose-Languages (GPLs)*, die für verschiedene Domänen nützlich sind, wie z. B. die Programmiersprache Java [GJS00].

GPLs sind in der Regel turing-vollständig und bieten viele Freiheiten, so dass heutzutage die meiste Software mit ihnen entwickelt wird. Dadurch ergibt sich eine sehr große Zahl an Anwendern, die den beträchtlichen Entwicklungsaufwand einer GPL hinsichtlich Zeit und

[1] Ein abgrenzbares Problemfeld und ein spezifischer Anwendungsbereich (für ein Computersystem) sind Kriterien einer *Anwendungsdomäne*. Somit bezeichnet der Begriff *Domäne* in DSL sowohl eine Anwendungs- als auch eine Wissensdomäne.

aufgewendeter Arbeitsleistung rechtfertigen. Als Nachteil einer GPL zeigt sich, dass ihre Anwender über eine hohe Qualifikation und Kenntnis der ihr zugrundeliegenden Konzepte und ihre Handhabung verfügen müssen. Besitzt die Domäne, auf welche die GPL angewendet werden soll, ebenfalls hinreichend komplexe Konzepte, so vervielfacht sich die Komplexität, die aus beidem resultiert. Eine Möglichkeit, diese Komplexität zu reduzieren, ist die Anwendung des Prinzipes der *separation of concerns*, beispielsweise indem man Funktionalität in Bibliotheken zu logischen Einheiten kapselt. Doch diese Bibliotheken werden mittels einer GPL erstellt und entsprechend erfordert ihre Nutzung ebenfalls spezielle Kenntnisse über die verwendete GPL. Folglich müssen sich die Anwender dieser Bibliotheken ebenfalls mit der GPL auskennen.

Oftmals muss man eine andere Zielgruppe von Anwendern berücksichtigen, bei denen das Wissen über die Domäne konzentriert ist, die sogenannten *Domänen-Experten*. Üblicherweise verfügt der Domänen-Experte nicht zusätzlich über fundierte Kenntnisse einer GPL. Abhilfe schafft hierbei, dem Domänen-Experten eine DSL zur Verfügung zu stellen, deren Sprachkonzepte er aus seiner Domäne und Arbeitsweise kennt. Durch Nutzung dieser DSL kann er dem Computer seine domänenspezifischen Informationen mitteilen.

Domain-Specific-Languages werden in *interne* und *externe DSLs* unterteilt. Eine *interne DSL* basiert auf einer GPL und definiert mit den dort verfügbaren Sprachmitteln eine eigene Syntax und Semantik (z. B. SLX). Sie wird deshalb auch *eingebettete DSL* genannt. Demgegenüber stehen *externe DSLs*, die eigene Parser erfordern und deren Syntax und Semantik keinen Bezug zu einer GPL aufweist (z. B. ExpL).

Wenn die Domäne z. B. *Schachspielen* ist, dann könnten Konzepte, wie *Figur*, *Spielfeld*, *Zug* und *Zeit* identifiziert werden. Eine mögliche Relation dieser Konzepte wäre, dass ein Zug eine regelkonforme Bewegung einer Figur innerhalb einer bestimmten Zeit auf dem Spielfeld ist. Typischerweise enthält eine Domäne auch etablierte Formen der Notation, wie im Schachspiel z. B. die *Algebraische Notation* (z. B. Lc4 – Läufer zieht nach Feld c4).

DSLs sind nicht neu: Beispielsweise kann man eines der ältesten, 1973/74 entwickelten UNIX-Werkzeuge, den *stream editor (sed)* als DSL für die Manipulation von Texten verstehen. Seit etwa einer Dekade „neu“ ist hingegen die Technologie der *metamodellbasierten* Sprachentwicklung von DSLs, die in dieser Arbeit verwendet und in Abschnitt 6.3 auf Seite 65 dargestellt wird.

6.2. Aspekte einer Sprache

Computersprachen, insbesondere Domain-Specific-Languages, müssen eine wohl definierte Struktur und eine klare Bedeutung besitzen, damit sie für einen Computer verständlich sind. Auf der anderen Seite muss ein Computer entsprechend programmiert werden, um Struktur und Bedeutung dieser Sprache verstehen zu können. Bei Domain-Specific-Languages spielt zudem eine große Rolle, dass sie menschenlesbar sind.[2]

[2] Somit helfen sie bei der Verständigung von Menschen und Computern. Martin Fowler bemerkt hierzu: „Any fool can write code that a computer can understand. Good programmers write code that humans can

Die folgenden Unterabschnitte beschäftigen sich mit den wichtigsten Aspekten einer Sprache und der hierfür notwendigen Terminologie, die aus der Dissertation von Scheidgen übernommen wurde [Sch09]. Eine erneute, kritische Begriffsbildung wird somit bewusst an dieser Stelle eingespart, da die hier vorliegende Arbeit sich nicht mit der Sprachforschung beschäftigt, sondern die Anwendbarkeit eines sprachzentrierten Ansatzes (und somit die Entwicklung von DSLs) für die Beschreibung von Experimentier-Workflows untersucht. Ergänzend werden an geeigneten Stellen Verweise zur Arbeit von Sadilek [Sad09] gegeben, der eine strenge, mengentheoretische Definition (auf Basis von [HR00]) von sprachbezogener Terminologie vornimmt.

In der formalen Sprachtheorie ist eine *Sprache*[3] eine Menge von *Sprachäußerungen* (*language utterances*) [MAKP88]. Scheidgen benennt diese Menge als *Sprachinstanzen*, da diese eine Klasse oder Gruppe von Elementen mit gemeinsamen Charakteristika bildet [Sch09]. Eine *Sprachinstanz* ist demnach ein Objekt mit wohl definierter Struktur und Bedeutung, und eine *Sprache* ist eine Menge von Sprachinstanzen [Sch09, S. 6]. Eine *Sprachbeschreibung* ist darauf aufbauend ein endliches Regelsystem, das die möglichen Instanzen einer Sprache beschreibt. Eine konkrete Methode der Sprachbeschreibung ist beispielsweise die Metamodellierung (siehe Abschnitt 6.3).

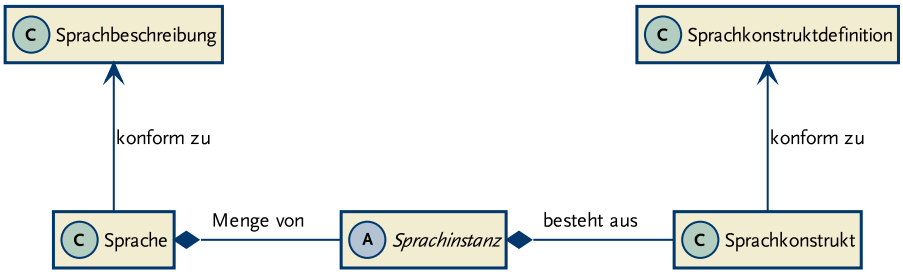


Abbildung 6.1.: Relation zwischen Sprachtermini (als UML-Klassendiagramm)

6.2.1. Syntax

Eine Sprachinstanz hat eine Struktur, die aus kleineren Bestandteilen, den *Sprachkonstrukten* besteht [Sch09, S. 7]. Sprachkonstrukte sind definiert durch *Sprachkonstruktdefinitionen*. Jedes Vorkommen eines Sprachkonstruktes innerhalb einer Sprachinstanz ist eine *Sprachkonstruktinstanz*, die durch die zugehörige Sprachkonstruktdefinition definiert ist. Dabei können Sprachkonstruktdefinitionen untereinander in Relation stehen, wobei ihre zugehörigen Sprachkonstruktinstanzen ebenfalls diese Relation besitzen (durch Instanziierung).

understand."

[3] Scheidgen [Sch09, S. 6] weist auf die verwirrende Mehrdeutigkeit des Begriffs Sprache und dessen Verwendung in der Informatik hin. Die von ihm gegebene Definition ist jedoch konsistent zur gebräuchlichsten Definition aus der formalen Sprachtheorie, nach der eine Sprache eine Menge von Worten über einem Alphabet ist [MAKP88]. Sprachinstanzen können somit als Worte über einem Alphabet verstanden werden.

Die *Syntax* einer Sprachinstanz ist die Struktur, die durch die verbundenen Sprachkonstruktinstanzen gebildet wird. Die *Syntaxbeschreibung* einer Sprache ist ein System von Sprachkonstruktdefinitionen, die die Struktur von möglichen Sprachinstanzen beschreiben.

Scheidgen weist darauf hin, dass die Konzepte Syntax und Sprachinstanz eng miteinander verbunden sind: Syntax meint die Struktur einer Instanz, die Sprachinstanz steht für sich selbst. Verwendet man z. B. kontextfreie Grammatiken als Sprachdefinitionsmethode, so besteht die Sprachinstanz aus einer Zeichenkette (*string*) von Terminalsymbolen. Die Syntax dieser Zeichenkette ist hingegen eine baumartige Struktur, die repräsentiert, wie die Zeichenkette aus den Grammatikregeln aufgebaut wurde.

In anderen Fällen, wie z. B. unter Verwendung einer metamodellbasierten Sprachdefinition, kann nicht sinnvoll zwischen Syntax und Sprachinstanz unterschieden werden. In diesem Fall hat die Sprachinstanz keine konkrete Form; die Instanz ist selbst eine Struktur, was die Definition für eine Syntax erfüllt. In metamodellbasierten Sprachdefinitionen wird die konkrete Form (bzw. die Repräsentation, siehe Abschnitt 6.2.2) der Sprachinstanz durch Notationen angegeben, die separat von der Sprache definiert werden.

Scheidgen merkt weiterhin an, dass manchmal die Begriffe *konkrete Syntax* und *abstrakte Syntax* verwendet werden, um zwischen (konkreten) Sprachinstanzen und ihrer (abstrakten) Syntax zu unterscheiden, beispielsweise im Falle von Sprachdefinitionen durch kontextfreie Grammatiken. Bei anderen Methoden der Sprachdefinition ändert sich die Bedeutung dieser Begriffe; aufgrund dieser Bindung an die Methode verzichtet Scheidgen auf diese Begriffe.

Sadilek hingegen verwendet beide Begriffe im Kontext von metamodellbasierten Sprachdefinitionen in seiner Dissertation [Sad09, S. 28 f.]. Dabei nutzt er einen mengentheoretischen Ansatz zur Sprachdefinition und präzisiert beispielsweise „abstrakte Syntax“ mit zusätzlichen Attributen, wie „eines Programmes/Modells“ oder „ein Programm/Modell in einer generischen, konkreten Syntax“.

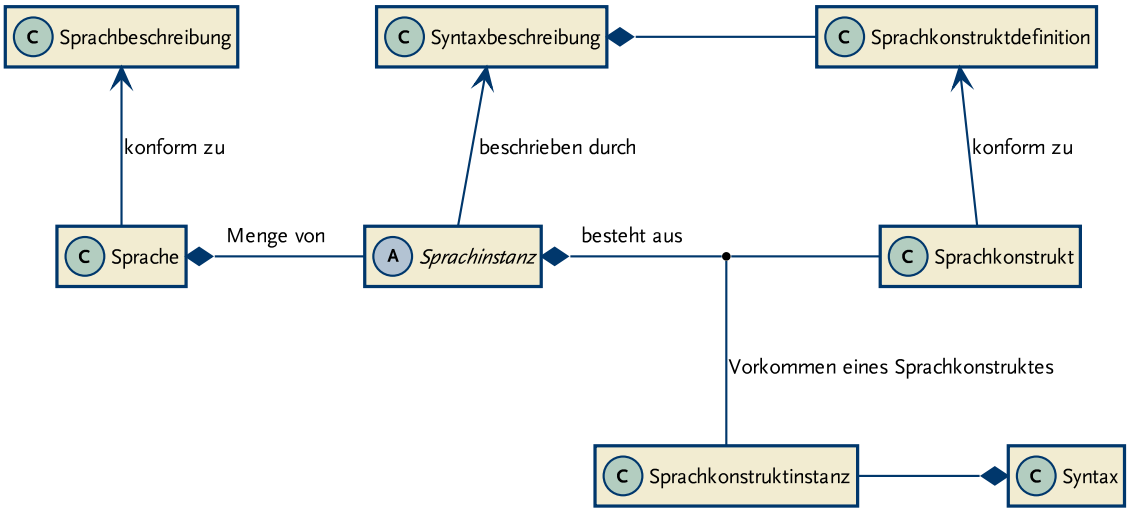


Abbildung 6.2.: Relation zwischen Sprachtermini unter Einbeziehung des Begriffes der Syntax (erweitert Abbildung 6.1 auf der vorhergehenden Seite)

6.2.2. Repräsentation

Sprachinstanzen können konkret oder abstrakt sein. Eine *konkrete Sprachinstanz* ist von Menschen erzeugbar, lesbar und für sie verständlich, wie z. B. ein Textfragment, ein Diagramm oder eine Tabelle. Eine *abstrakte Sprachinstanz* hingegen ist lediglich eine Struktur, die nicht notwendigerweise eine (solche) physische Form hat.

Eine Sprachinstanz kann durch eine andere Sprachinstanz (einer anderen Sprache) repräsentiert werden, die somit als eine *Repräsentation* dieser ersten Sprachinstanz bezeichnet wird. Der Begriff der Notation ist daraus wie folgt abgeleitet: Eine Sprache mit Instanzen, die Repräsentationen einer anderen (der *notierten*) Sprache sind, wird als *Notationssprache* für die notierte Sprache benannt. Entsprechend wird eine Notationssprache in Kombination mit einer Abbildung von der Notationssprache zu der notierten Sprache als *(Sprach-)Notation* benannt.

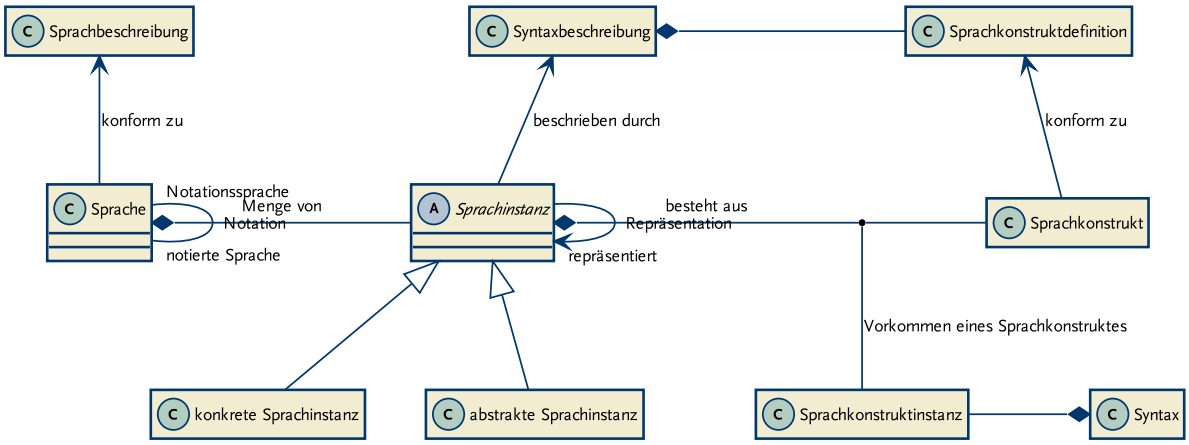


Abbildung 6.3.: Relation zwischen Sprachtermini unter Einbeziehung der Begriffe Repräsentation und Notation (erweitert Abbildung 6.2)

6.2.3. Semantik

Wie in der Einleitung dieses Kapitels 6 ausgeführt wurde, ist die Aufgabe einer Sprache die Übermittlung von Information. Mit den eingeführten Begriffen soll dies präzisiert werden: Eine Sprachinstanz hat eine Bedeutung, um Informationen ausdrücken zu können. Die Information stammt dabei aus einer *semantischen Domäne*, welche die Bedeutung der zugehörigen Sprachinstanzen bestimmt.

Die Bedeutung einer Sprachinstanz ist die *Semantik* (z. B. handelt es sich bei einem Sprachkonstrukt um einen Ausdruck, eine Anweisung oder ein Kommando). Eine *semantische Abbildung* ist die Relation zwischen einer Sprache und einer semantischen Domäne. Die semantische Abbildung zusammen mit der semantischen Domäne bildet die *Sprachsemantik*.

Für eine Sprache können multiple semantische Abbildungen und semantische Domänen verwendet werden, so dass diese Sprache verschiedene Sprachsemantiken erhält. Sadilek gibt

hierfür ein Beispiel [Sad09, S. 32] einer einfachen Sprache $\mathcal{L}$ für arithmetische Ausdrücke E, die beschreiben, wie Zahlen addiert und multipliziert werden. Sei die (abstrakte) Syntax $\mathcal{A}$ dieser Sprache $\mathcal{L}$ durch folgende Grammatik $\mathcal{G}$ definiert:

$$E ::= \text{<numerisches Literal>} \,|E + E|E * E$$

Die Semantik der Sprache $\mathcal{L}$ ist die Auswertung von der Grammatik $\mathcal{G}$ entsprechenden Ausdrücken. Entsprechend ist die zugehörige, semantische Domäne $\mathcal{S}$ die Menge der reellen Zahlen $\mathbb{R}$, es gilt also $\mathcal{S} = \mathbb{R}$. Die semantische Abbildung $\mathcal{M}$ weist jedem Ausdruck eine reelle Zahl zu: $\mathcal{M} : \mathcal{A} \to \mathbb{R}$. Beispielsweise: $\mathcal{M}(1 + 1) = 2$.

Entsprechend der hier gegebenen Definition einer Sprache kann man sich für das gegebene Beispiel eine zweite Semantik vorstellen.[4] Angenommen, man würde, wie manche Kinder, die Finger als Hilfsmittel zur Berechnung einsetzen. So könnte man in diesem Fall zu jedem Ausdruck die Anzahl an Händen angeben, die für die Berechnung des Ausdruckes hilfsweise nötig sind, entsprechend gilt $\mathcal{S} = \mathbb{N}$. Beispiele hierfür sind: $\mathcal{M}(1 + 1) = 1$, $\mathcal{M}(1 + 4) = 1$ und $\mathcal{M}(6 + 2) = 2$.

Abbildung 6.4 visualisiert zusammenfassend, wie die hier vorgestellten Konzepte Sprache, Notation, Repräsentation, Semantik und semantische Domäne zusammenhängen. Dabei wird das Konzept der Relation zweimal in Form einer Abbildung verwendet. Einmal als semantische Abbildung zwischen Sprache und semantischer Domäne und einmal als Abbildung zwischen Notationssprache und Sprache. Beide Relationen sind n:m stellig.

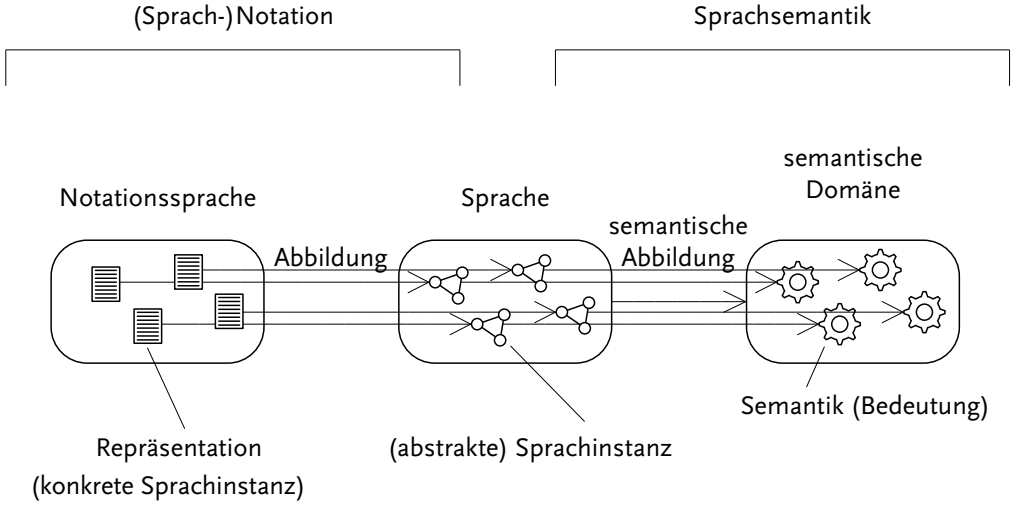


Abbildung 6.4.: Aspekte einer Sprachinstanz: Repräsentation und Semantik (basierend auf [Sch09, S. 9])

[4] Auf eine formale, vollständige Definition soll an dieser Stelle verzichtet werden.

6.3. Metamodellbasierte Sprachentwicklung

Die Entwicklung bzw. die Definition von Sprachen erfolgt mit Hilfe verschiedener Methoden für die Beschreibung der verschiedenen Sprachaspekte. Die Methode der (kontextfreien) Grammatik zur Definition von Repräsentation und Syntax wurde bereits in Abschnitt 6.2.1 angesprochen. In diesem Abschnitt 6.3 werden alternativ existierende Methoden auf der Basis von Modellen beschrieben.

Als *Sprachwerkzeuge* sollen dabei im weiteren Sinne alle (Software-)Werkzeuge verstanden werden, die zur Definition von Syntax, Notation und Semantik einer Sprachinstanz und ihrer Verwendung dienlich sind. Die Idee, solche Sprachwerkzeuge partiell oder komplett aus den Sprachbeschreibungen zu generieren, existiert seit den ersten (*high level*) Programmiersprachen. In den Anfängen waren solche Programmiersprachen rein textuell notiert, und es gab keine Unterscheidung (die auch nicht notwendig war) zwischen Repräsentation und Sprache.

In den 1960er Jahren entstand die Backus-Naur-*Form (BNF)* als eine kompakte, formale Notation zur Darstellung kontextfreier Grammatiken [Knu64]. Mit BNF und kontextfreien Grammatiken war es möglich, die Notation einer Sprache zu beschreiben, also die Menge aller durch die Grammatik erzeugbaren Zeichenketten (*strings*, i. A. die Sprachinstanzen), und ihre Syntax (i. A. die Sprachkonstrukte). Allerdings fehlte bei dieser Methode die Möglichkeit, Beschränkungen (z. B. für kontextabhängige Syntax) und Semantik zu beschreiben. Scheidgen weist auf zwei Lösungsansätze hierfür hin [Sch09, S. 13]: Attributgrammatiken (entwickelt von Knuth [Knu90]) und Graphgrammatiken. Beide Ansätze konnten sich in der Praxis eher wenig durchsetzen. Zu den Gründen spekuliert Scheidgen, dass Attribut- und Graphgrammatiken als rigorose mathematische Formalismen nicht leicht zu meistern und als geschlossene Formalismen schwer erweiterbar sind.

Modelware und Metamodell

Demgegenüber steht die objektorientierte Modellierung als kohärente, etablierte Methode zur Beschreibung von Systemen. Als Standard für eine Sprache zur objektorientierten Modellierung hat sich die *Unified Modeling Language (UML)* [20007b] etabliert, wobei sich insbesondere die in der UML definierten Klassendiagramme als Notationsform auf breiter Front durchgesetzt haben (wobei Konzepte wie *Klasse* und *Relation zwischen Klassen* verwendet werden). Die UML und die um sie herum existierenden, modellgetriebenen Techniken bilden einen Technologieraum[5] (*technological space*), der in der Literatur als *modelware* referenziert wird (beispielsweise Synonym für modellgetriebene Architektur in [WK06]).

Im Modelware-Technologieraum werden Sprachaspekte (z. B. die Syntax) nicht mit Grammatiken beschrieben, sondern mit objektorientierten Modellierungsmethoden, die im Ergebnis zu einem objektorientierten Modell führen. Dieses Modell repräsentiert eine objektorientierte Datenstruktur (im Speicher eines Computers), deren mögliche bzw. konforme Struktu-

[5] Ein Technologieraum ist nach Kurtev et al. [KBA02] ein „working context with a set of associated concepts, body of knowledge, tools, required skills, and possibilities".

ren beschrieben sein müssen, damit man mit solchen Modellen arbeiten kann. Diese konformen Strukturen sind mittels eines weiteren Modells beschrieben – somit handelt es sich um ein Modell eines Modells, das *Metamodell*[6] genannt wird. Diese weit gefasste Definition eines Metamodells könnte man auf viele Arten von Strukturbeschreibungen anwenden, wie beispielsweise auf Schemata für Datenbanken oder für Schemata der *Extensible Markup Language (XML)*. Ohne auf die im Detail bestehenden Unterschiede einzugehen, soll der Begriff Metamodell in dieser Arbeit, wie auch in der Literatur üblich, nur im Kontext des Modelware-Technologieraumes angewendet werden.[7] Im Modelware-Kontext werden Metamodelle in Form von Klassendiagrammen notiert. Als Notationssprachen hierfür haben sich die UML(-Klassendiagramme) und *Ecore* durchgesetzt (auf Ecore wird in Abschnitt 9.1.2 eingegangen). Eine Sprachbeschreibung in Form eines Metamodells wird als *Sprachmetamodell* bezeichnet.

Vorteile

Scheidgen führt den Begriff *Objektorientierte Meta-Modellierung (OOMM)* für die Beschreibung von Computersprachen unter Verwendung von objektorientierten Modellierungsmitteln ein [Sch09, S. 14]. Darunter fällt die zuvor dargestellte Beschreibung von Sprachkonstrukten mit Klassen und Relationen zwischen Klassen. Scheidgen definiert OOMM weitreichender, so dass dieser Begriff nicht nur die Sprachbeschreibung mittels Klassendiagrammen umfasst, sondern auch die Beschreibung von Repräsentation und Semantik einschließt.

OOMM weist eine Menge von Vorteilen gegenüber grammatikbasierten Sprachbeschreibungen auf (siehe [Sch09, S. 15]):

- OOMM ist einfach handhabbar für Sprachentwickler, die gewöhnlich bereits mit objektorientierten Techniken vertraut sind.
- OOMM erlaubt es, die Komplexität einer Sprache leichter mit etablierten, objektorientierten Techniken auszudrücken, wie beispielsweise: Abstraktion, Wiederverwendung, Modularisierung oder Vererbung.
- OOMM verwaltet Graphen, wodurch alle Arten von Sprachinstanzen unterstützt werden.
- OOMM bietet eine klare Trennung von Sprachinstanzen und ihren Repräsentationen, wodurch alle Arten von Notationsformen unterstützt werden.
- OOMM entspricht dem vorherrschenden, objektorientierten Programmierparadigma.

Sprachkopplung Diese Vorteile auf technischer Ebene begründen bereits den Einsatz von OOMM in dieser Arbeit. Auf der Basis von OOMM lässt sich zudem eine *Sprachkopplung*

[6] Die Vorsilbe *meta* stammt aus dem Griechischen von $\mu\varepsilon\tau\acute{\alpha}$, für „zwischen, inmitten, nach, nachher, später“ [Dud11]. Im Kontext von Metamodellen kann es übersetzt werden mit „über“ oder „übergeordnet“.

[7] Somit ist der Begriff *Metamodell* in dieser Arbeit synonym zum Begriff *sprachbasiertes Metamodell*.

umsetzen, was bedeutet, dass verschiedene Sprachbeschreibungen mit Hilfe ihrer Sprachmetamodelle verbunden werden. Eine solche Sprachkopplung kann technisch verschiedenartig realisiert werden, verlangt aber i. A. ein gemeinsames Metametamodell. In dieser Arbeit ist ein pragmatischer Ansatz gewählt worden, bei dem die verschiedenen Sprachmetamodelle zu einem neuen, gemeinsamen Sprachmetamodell zusammengesetzt werden.[8]

6.4. Relation zur Workflow-Terminologie

In dieser Arbeit werden Methoden der metamodellbasierten, domänenspezifischen Sprachentwicklung zur Formalisierung von Experimentier-Workflows angewendet (die somit die Domäne bilden). Entsprechend ist in diesem Abschnitt die Relation wichtiger Termini aus der Workflow-Begriffswelt und der Begriffswelt der domänenspezifischen Sprachen dargestellt (eine Zusammenfassung bietet Tabelle 6.1 auf der folgenden Seite). Die OMG definiert innherhalb ihres Standards *Meta Object Facility (MOF)* eine vierstufige Hierarchie von Meta-Ebenen (M0 bis M3), die in der folgenden Darstellung zur Orientierung genutzt wird [20006a].

In der metamodellbasierten Sprachentwicklung liegt die Sprachbeschreibung in Form eines Metamodells vor, das eine Menge von Regeln darstellt (Metamodellschicht M2 in Tabelle 6.1). Das Metamodell definiert die Menge der zur Sprachbeschreibung konformen Sprachinstanzen (Modellschicht M1). In der Workflow-Begriffswelt ist die Workflow-Definition analog zur Sprachinstanz. Durch die Sprachinstanz wird die darin enthaltene Information formalisiert (konform zu den Regeln in der Sprachbeschreibung), die an den Computer übermittelt werden soll. Die Workflow-Definition formalisiert den ihr zugrundeliegenden Prozess. In der Workflow-Begriffswelt gibt es keine Entsprechung für die Sprachbeschreibung, weshalb in dieser Arbeit der analoge Begriff *Workflow-Beschreibung* eingeführt wird, der ein Regelsystem (z. B. in Form eines Metamodells) bezeichnet und somit die Menge an den hierzu konformen Workflow-Definitionen festlegt.

Eine Workflow-Definition wird durch ein Workflow-Management-System instanziiert, wodurch eine Menge von konformen Workflow-Instanzen erzeugt wird, die zur Laufzeit ausgeführt werden (Datenschicht M0). Eine entsprechende Bezeichnung in der DSL-Begriffswelt zu finden, gelingt nicht, so dass für die Repräsentation[9] einer Sprachinstanz innerhalb von Sprachwerkzeugen die Bezeichnung *Sprachinstanz-Laufzeitrepräsentation* eingeführt wird. Die Begriffe Workflow-Management-System und Sprachwerkzeuge sind analog, da sie jeweils sehr ähnliche Funktionen anbieten, die in der Modellierung, Instanziierung und Verwaltung bestehen. Zu unterscheiden ist lediglich der jeweilige Gegenstand: Ein Workflow-Management-System unterstützt bei der Modellierung des Prozesses (resultiert in der Workflow-Definition) mit der Bereitstellung von Werkzeugen, instanziiert die Workflow-Definition (resultiert in einer oder mehrerer Workflow-Instanzen) und verwaltet die Workflow-Instanzen (als Workflow-Lebenszyklus). Sprachwerkzeuge sind im weiteren Sinne alle (Software-)Werkzeuge, die zur

[8] Andere Formen von Sprachkopplung können über Referenzen zwischen Sprachmetamodellen erfolgen. Der gewählte, pragmatische Ansatz verursachte die wenigsten Probleme bei den verwendeten Sprachwerkzeugen.

[9] Eine solche Repräsentation kann beispielsweise ein *Abstract-Syntax-Tree (AST)* sein.

Definition von Syntax, Notation und Semantik einer Sprachinstanz und ihrer Verwendung dienlich sind. Fügt man die eigentliche Definition (Sprachinstanz bzw. Workflow-Definition) den Begriffen Workflow-Management-System und Sprachwerkzeuge hinzu, so erhält man als Gesamtheit *Workflow-System* und *Sprachsystem* (neu eingeführter Begriff).

Die MOF-Ebene M3 wird repräsentiert durch ein *Metametamodell*, welches das Metamodell in Ebene M2 definiert. Metametamodelle nach dem MOF-Standard können sich selbst beschreiben (wodurch weitere, höhere Ebenen unnötig sind) und werden als *selbstreferentielle Metametamodelle* bezeichnet. Durch diesen hierarchischen Aufbau ist jede Ebene formal definiert durch die nächst höhere (oder sich selbst im Falle von M3). Die letzte Spalte von Tabelle 6.1 zeigt die Begriffe in der Anwendung auf die in dieser Arbeit entwickelten Sprache ExpL (und somit auf Experimentier-Workflows). Weil ExpL eine Beschreibungssprache für (Experimentier-)Workflows ist, wurden die Workflow-Termini verwendet.

Ebene	**Workflow-Terminus**	**Sprach-Terminus**	**Erklärung**	**in dieser Arbeit**
M3			Metametamodell, definiert ein Metamodell	(Metametamodell Ecore)
M2	*Workflow-Beschreibung*	Sprach-beschreibung	Menge von Regeln (z. B. Metamodell, Grammatik)	ExpL-Sprach-metamodell
M1	Workflow-Definition	Sprachinstanz	regelkonforme Formalisierung von Prozess bzw. Information	ExpL-Workflow-Definition
M0	Workflow-Instanz	*Sprachinstanz-Laufzeitrepräsentation*	Laufzeitebene	ExpL-Workflow-Instanz
	Workflow-Management-System	Sprachwerkzeuge	Management-System für Modellierung, Instanziierung	ExpL-Workflow-Management-System
	Workflow-System	*Sprachsystem*	Gesamtheit von Definition, Instanz und Management-System	ExpL-Workflow-System

Tabelle 6.1.: Relationen zwischen wichtigen Termini von Workflows und domänenspezifischen Sprachen, ergänzt mit neu eingeführten Begriffen (*hervorgehoben*)

Teil II.

Von Experimentier-Prozessen zu Experimentier-Workflows

Wer als Werkzeug nur einen Hammer hat, sieht in jedem Problem einen Nagel.

Paul Watzlawick

7. Experimentier-Prozesse in Fallstudien und Anforderungen

Kapitel 7 beschreibt den Experimentier-Prozess in jeder der drei durchgeführten Fallstudien, wie er traditionell durchgeführt wurde, ohne den Ansatz aus dieser Arbeit zu verwenden. Dabei wird jede Fallstudien in diesem Kapitel durch folgende, einheitliche Struktur dargestellt:

Gegenstand der Untersuchung Erläutert die Art und Modellklasse des zu untersuchenden Modells (MUX). Jedes dieser MUX existiert bereits und ist ein dynamisches Modell. Zusätzlich werden auch das Untersuchungsziel und der domänenspezifische Kontext in diesem Abschnitt dargestellt.

Traditioneller Experimentier-Prozess Erklärt informal den Experimentier-Prozess zu Beginn der Fallstudie und wie er durchgeführt wurde, ohne den Ansatz aus dieser Arbeit zu verwenden.

Probleme und Anforderungen Beschreibt die erkannten Probleme und die daraus abgeleiteten Anforderungen, sowohl an eine formale Beschreibung des Experimentier-Prozesses, als auch an ein Computersystem, das diese formale Beschreibung zur Automatisierung verwendet.

Die Anwendung des in dieser Arbeit entwickelten Ansatzes in den drei Fallstudien (u. a. in Form von ExpL-Workflows) zeigt Kapitel 10. Teile der Implementierung der Experimentier-Workflows sind in Anhang C dargelegt.

7.1. Geographie: SLEUTH – Ein Modell des urbanen Landnutzungswandels

Die SLEUTH-Fallstudie entstand in einer Kooperation zwischen Geomatikern und Mitgliedern des Informatik-Graduiertenkollegs METRIK.[1] Ein langfristiges Ziel der Geomatiker ist es, verschiedene Modellierungstechniken[2] für die Beschreibung des Landnutzungswandels (insbesondere der Urbanisierung) in unterschiedlichen Regionen zu vergleichen. In der

[1] Die Kooperationspartner in der SLEUTH-Fallstudie waren: Prof. Dr. Tobia Lakes (Geographisches Institut der Humboldt-Universität zu Berlin, Abteilung Geomatik), Carsten Krüger (ebenso Abteilung Geomatik, später in METRIK) sowie Falko Theisselmann und Frank Kühnlenz (beide in METRIK).

[2] Solche Modellierungstechniken sind beispielsweise künstliche neuronale Netze [PAM06], *Support-Vector-Machines (SVMs)* [HXTW09], Agenten-basierte Systeme [GSB+08] oder Zelluläre Automaten [MC11].

SLEUTH-Fallstudie wurde eine etablierte Modellklasse betrachtet, die den Formalismus *Zellulärer Automat* [MC11] verwendet. Diese etablierte Modellklasse wurde re-implementiert, erweitert und in ihrer Ausdruckskraft mit der ursprünglichen Ausgangsmodellklasse verglichen.

7.1.1. Gegenstand der Untersuchung

SLEUTH-Modellklasse

In der SLEUTH-Fallstudie wurde das von Clarke etablierte[3] Modell zur Beschreibung von Landnutzungswandel (hinsichtlich Urbanisierung) untersucht, das den Namen SLEUTH (*Slope, Land-cover, Exclusion, Urbanization, Transportation, Hillshade*) trägt [Cla08, CHG97]. Clarke trifft keine sprachliche Unterscheidung zwischen Modell und Modellklasse: Sein „Modell“ ist derart parametrisiert, dass eine Anpassung bzw. Kalibrierung auf die Spezifik in einer bestimmten geographischen Region möglich wird. Somit bezeichnet der Modellbegriff von Clarke eine Modellklasse. Der Begriff SLEUTH wird in dieser Arbeit entsprechend dem einer Modellklasse verwendet. In der SLEUTH-Fallstudie wurde ausschließlich die Region des Großraums Tirana in Albanien betrachtet, so dass der Begriff *SLEUTH-Modell* als für diese Region kalibriert verwendet wird.

Die SLEUTH-Modellklasse ist zeitdiskret, dynamisch und basiert auf dem Formalismus *Zellulärer Automat*, der hier mit georäumlichen Eigenschaften[4] erweitert ist. Seine Zellen bilden ein Raster im zweidimensionalen Raum. Die SLEUTH-Modellklasse beschreibt Verhalten derart, dass für jede Zelle ein Entwicklungspotential hinsichtlich Urbanisierung berechnet wird (abhängig von Faktoren wie beispielsweise der Gradiente oder der Nähe zu Straßen). Dadurch kann der Urbanisierungszustand einer Zelle von „nicht-urbanisiert“ zu „urbanisiert“ wechseln.

Die Aktualisierung des Zustandes einer Zelle ist definiert durch eine Menge von Transitionsregeln und findet in diskreten Zeitschritten statt. Die Transitionsregeln können auf den eigenen Zustand der Zelle, sowie auf die Zellen in ihrer deterministischen Moore-Nachbarschaft[5] zugreifen. Im Detail existieren vier Transitionsregeln, welche die Zustandsänderung einer Zelle festlegen [CHG96, CHG97]:

1. Spontanes Wachstum (*spontaneous growth*): Eine zufällig ausgewählte Zelle wird urbanisiert (Urbanisierungszustand wechselt auf „urbanisiert“), wenn sie in der Moore-Nachbarschaft einer bereits urbanisierten Zelle liegt.

[3] Die Webseite des Projektes [Cla11] zeigt gegenwärtig 32 Regionen weltweit, die mittels SLEUTH innerhalb der letzten Dekade untersucht wurden; Clarke selbst verweist auf über 100 Städte und Regionen [Cla08].

[4] Georäumliche Eigenschaften sind in diesem Kontext: Fehlende Invarianz bezüglich georäumlicher Umplatzierung und das Vorhandensein von georäumlichen Repräsentationen und Konzepten.

[5] Die Moore-Nachbarschaft (auch 8er-Nachbarschaft bezeichnet) ist eine nach Edward F. Moore benannte Nachbarschaftsbeziehung in einem quadratischen Raster. Alle Flächen, welche mindestens eine Ecke mit der Basisfläche gemeinsam haben, gelten als Nachbarn.

2. Flaches Gebiet (niedriger Wert der Gradiente) ist eine gute Bedingung für Urbanisierung, wodurch Zellen (wahrscheinlichkeitsabhängig) auch urbanisiert werden können, wenn sie nicht in der Nachbarschaft von urbanisierten Zellen liegen (*diffusive growth*).
3. Wachstumszentren (z. B. Städte) erhöhen die Wahrscheinlichkeit für urbanes Wachstum an ihren Rändern (*organic growth*).
4. Der Ausbau von Straßen fördert das urbane Wachstum entlang der neu entstandenen Transportwege (*road influenced growth*).

Zusätzlich zu diesen vier Transitionsregeln existieren sogenannte *self-modification growth rules*, die unkontrolliertes, exponentielles Wachstum und Schrumpfen verhindern, indem Grenzwerte für minimales und maximales Gesamtwachstum parametrisiert werden [CHG96].

Die SLEUTH-Modellklasse besteht aus zwei eng miteinander verwobenen Modellklassen: Dem *Urban Growth Model (UGM)*, das den Wandel der Urbanisierung modelliert und dem *Deltatron-Modell* (ein *land cover model*), das die natürlichen und künstlichen Anlagen auf der Landoberfläche beschreibt.[6] In dieser Fallstudie besteht keine Notwendigkeit, bezüglich dieser Teilmodellklassen zu unterscheiden, weshalb weiterhin der Oberbegriff „SLEUTH(-Modellklasse)“ verwendet wird (für weitere Details wird auf [Cla08] verwiesen).

Re-Implementierung und Erweiterung in der SLEUTH*P-Modellklasse

Die SLEUTH-Modellklasse benötigt eine Menge von gerasterten Eingabedatensätzen, die spezifisch für die Zielregion sind: Gradiente, urbane Landnutzung, Verkehr und auszuschließende Bereiche (siehe Abbildung 10.1 auf Seite 144). Für die zu untersuchende Zielregion des Großraums Tirana in Albanien standen lediglich Landnutzungsdaten für drei von fünf benötigten Jahren zur Verfügung. Aus diesem Grund war eine Modifikation der Ausgangsmodellklasse notwendig, um der vorhandenen, beschränkten Datenlage zu genügen [LTK+12].

Theisselmann nahm eine Analyse der SLEUTH-Modellklasse in der von Clarke vorliegenden C-Implementierung vor [TKK+10]. Folgende Gründe führten zu einer kompletten Re-Implementierung:

- Der Quellcode der SLEUTH-Implementierung ist über mehr als eine Dekade „historisch gewachsen“, wodurch sich der Aufwand für das Verständnis des Quellcodes signifikant erhöht. Dieser Aufwand besteht zudem für jeden Anwender erneut, der eine Erweiterung oder Anpassung der Modellklasse vornehmen möchte.
- UGM und Deltatron-Modell sind im Quellcode kaum voneinander zu unterscheiden.
- Die Konzepte eines Zellulären Automaten, wie z. B. Zelle oder Transitionsregel, sind im Quellcode strukturell schwer erkennbar. Dies ist dem Umstand geschuldet, dass eine auf diese Konzepte ausgerichtete Strukturierung durch die Verwendung einer General-Purpose-Language wie C nicht gefördert wird. Der hierfür notwendige Aufwand obliegt dem Entwickler, der bewusste Entscheidungen treffen muss, welche Wahl der

[6] Die Namensgebung von Clarke für UGM und Deltatron-Modell wird beibehalten, obwohl es sich ebenfalls um Modellklassen handelt.

Implementierungsart in C für ein Konzept des Zellulären Automaten am geeignetsten ist. In der SLEUTH-Implementierung gelingt dies oftmals nicht: Globale Felder bilden beispielsweise den Zustand *aller* Zellen ab; eine lokale Zelle zu betrachten wird dadurch erschwert. Verständlichkeit und Nachvollziehbarkeit werden behindert.

- Die massive Verwendung von globalen Feldern und Funktionen führt zu Seiteneffekten, die oftmals ungewollt sind und Fehler in den Modellen verursachen.

Die Re-Implementierung der SLEUTH-Modellklasse wurde von Theisselmann konform zu der von ihm entwickelten DSL *Environmental-Cellular-Automata-Language (ECAL)* vorgenommen [TKK+10, TDF09]. und wird als SLEUTH* bezeichnet. SLEUTH* wurde um zusätzliche Statistikmaße erweitert und ist implementiert unter der Verwendung von Java.

In der Fallstudie wurde zusätzlich untersucht, inwiefern die Verwendung von Populationsdaten die Aussagekraft der Modelle verbessern können. Auf der Basis dieser Idee implementierte Theisselmann die Erweiterung von SLEUTH*, welche als SLEUTH*P (SLEUTH* mit Populationstreiber) benannt ist. Zur leichteren Unterscheidung wird folgende Nomenklatur eingeführt: SLEUTH(*P) bezeichnet alle drei Modellklassen und SLEUTH*(P) verweist auf die beiden Modellklassen SLEUTH* und SLEUTH*P.

Abbildung 7.1.: Modellklassen SLEUTH, SLEUTH*, SLEUTH*P: Spezifikationssprachen und Modifikationen

7.1.2. Traditioneller Experimentier-Prozess

Das Verhalten der Transitionsregeln des in der SLEUTH-Modellklasse modellierten Zellulären Automaten wird durch fünf Koeffizienten gesteuert: *dispersion*, *breed*, *spread*, *slope* und *road gravity* [Cla08]. In der *Kalibrierungsphase* (SLEUTH-Modellmodus *calibrate*) wird die optimale Wertebelegung dieser Koeffizienten für eine bestimmte Region ermittelt und dadurch ein SLEUTH-Modell erstellt. Die optimale Wertebelegung dieser Koeffizienten wird durch zufällige Variation der Modelleingabeparameterbelegung in mehreren Kalibrierungsläufen ermittelt (Monte-Carlo-Studie[7]). Pro Kalibrierungslauf wird hierfür aus regionenspezifischen Eingabedatensätzen empirisches Datenmaterial verwendet, das die Urbanisierung jeweils *eines* vergangenen Jahres beschreibt (die sogenannten *Referenzdaten*).

Ein Koeffizientensatz ist aus der Menge der untersuchten Koeffizientensätze dann optimal, wenn die (durch Modellausführung produzierten) zugehörigen Modellbeobachtungen den

[7] Die *Monte-Carlo-Studie* ist ein numerisches Lösungsverfahren aus der Stochastik, das auf der Basis von sehr häufig durchgeführten Zufallsexperimenten beruht.

Referenzdaten am besten entsprechen. Diese Entsprechung wird durch bestimmte Statistikmaße beurteilt, von denen der Lee-Sallee-Index[8] am wichtigsten ist [CHG96].

Nach einer erfolgreichen Kalibrierungsphase kann durch die Prognosephase (Modellmodus *predict*) ein zukünftiger, urbaner Landnutzungswandel für die Zielregion prognostiziert werden. In der durchgeführten Fallstudie wurde ausschließlich die Kalibrierungsphase betrachtet. Abbildung 7.2 visualisiert den traditionellen Experimentier-Prozess mit dem SLEUTH-Modell, wobei alle Aktivitäten manuell erfolgen, mit Ausnahme der Ausführung der Kalibrierungsläufe.

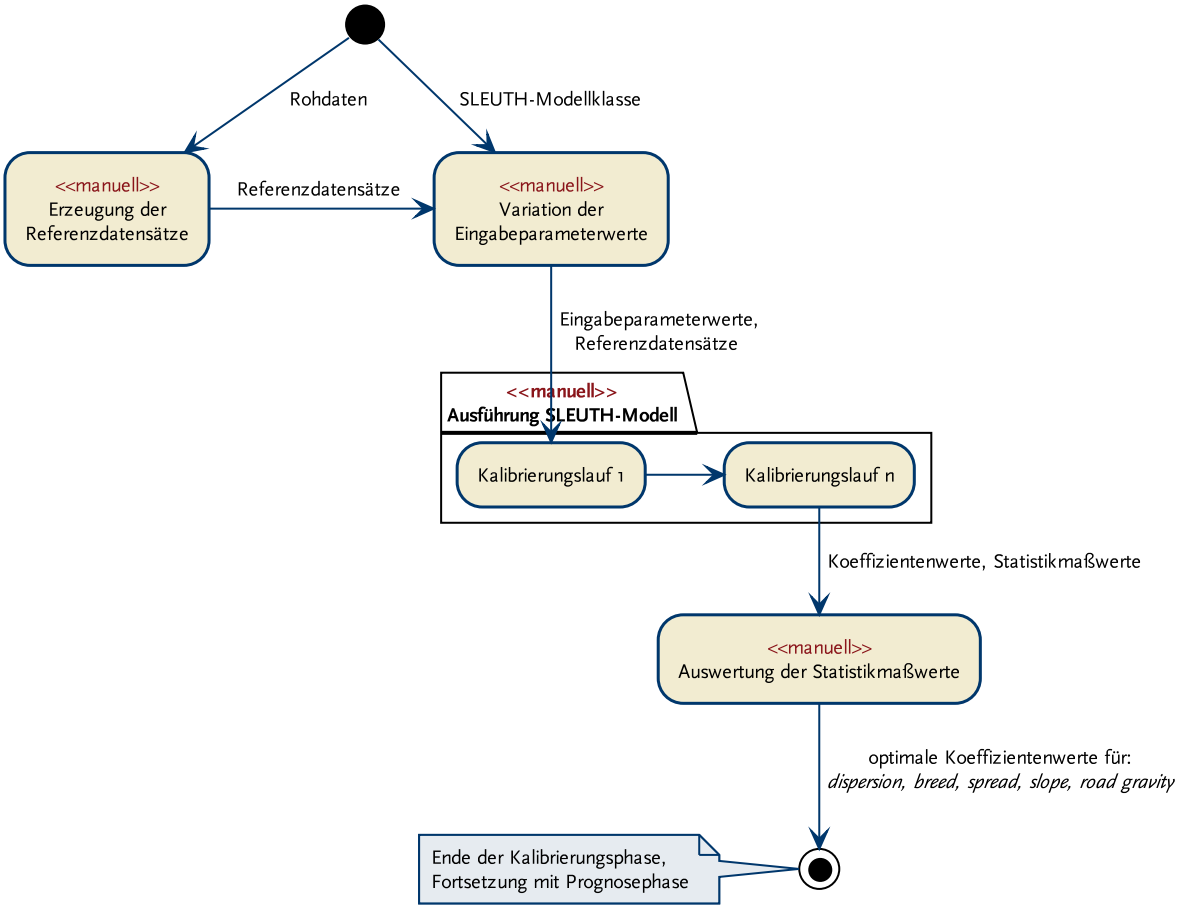


Abbildung 7.2.: Traditioneller Experimentier-Prozess mit SLEUTH: Kalibrierungsphase (als UML-Aktivitätsdiagramm)

[8] Der Lee-Sallee-Index ist ein georäumliches Ähnlichkeitsmaß zwischen dem empirisch beobachteten und dem vorhergesagten Landnutzungswandel. Er ist der Quotient aus den addierten Flächen (*union*) und der Schnittfläche (*intersection*) von Realität und Vorhersage [CHG96].

7.1.3. Probleme und Anforderungen

Probleme

Eine Analyse des traditionellen SLEUTH-Experimentier-Prozesses ergab folgende Probleme:

- Es wird keine Werkzeugunterstützung zur computerbasierten Automatisierung des Prozesses verwendet (und durch die SLEUTH-Implementierung auch keine angeboten).
- Verschiedene Aspekte des Experimentierens sind in der SLEUTH-Implementierung unübersichtlich miteinander verwoben:
  - Die Ablaufsteuerung der Phasen (Kalibrierung und Prognose) wird durch das Modell selbst durchgeführt. Somit ist ein Monitoring (Beobachten von Fortschritt, Zwischenergebnissen, Fehlern, usw.) nur durch das Modell selbst möglich, wofür es modifiziert werden muss.
  - Die Visualisierung des prognostizierten, urbanen Landnutzungswandels wird ebenfalls durch das SLEUTH-Modell selbst durchgeführt. Eine Änderung, beispielsweise der Farbgebung, bedingt eine erneute, potentiell zeitaufwendige Ausführung des Modells innerhalb des Experimentier-Prozesses.

Anforderungen

An ein Computersystem, das den Experimentier-Prozess in der SLEUTH-Fallstudie unterstützen könnte, gibt es folgende Anforderungen:

- Erzeugen und Ausführen des MUX-Programmes:
  - auf der Basis von Java (für die Modellklassen SLEUTH* und SLEUTH*P)[9],
  - Übergabe von Werten für Modelleingabe- und Laufzeitparameter,
  - sequentielle, nicht-interaktive und überwachte Ausführung,
  - Einlesen der Modellergebnisse.
- Variation von Modelleingabeparameterbelegungen mit Werten äquidistanter Schrittweite innerhalb eines Bereiches,
- Unterstützung der Auswertung von Modellergebnissen mit georäumlichem Bezug (z. B. unter Verwendung eines Geoinformationssystems).

[9] Die unveränderte SLEUTH-Modellklasse in der Sprache C war nicht Gegenstand des Experimentierens in der SLEUTH-Fallstudie.

7.2. Physik: Entwicklung optischer Nanostrukturen

Die Nano-Fallstudie ist eine Kooperation zwischen der Physik und dem Informatik-Graduiertenkolleg METRIK.[10] Das Ziel dieser Kooperation besteht in der Erstellung und Anwendung einer domänenspezifischen Sprache zur strukturellen Beschreibung von optischen Nanostrukturen (Nano-DSL) und dem Experimentieren mit den in dieser Sprache erstellten Strukturmodellen [WSKF11, Sch11].

7.2.1. Gegenstand der Untersuchung

Optische Nanostrukturen und Photonische Kristalle

Optische Nanostrukturen sind kleiner als die Wellenlänge des Lichtes. Von besonderem Interesse sind periodische, optische Nanostrukturen, die sogenannten *photonischen Kristalle*[11]. Photonische Kristalle können u. a. durch Beugung und Interferenz auf die Bewegung von Lichtphotonen einwirken. Dadurch weisen sie Eigenschaften auf, die Photonen ähnlich beeinflussen wie Halbleiter die Bewegung von Elektronen. Man spricht deshalb auch von „*optischen Halbleitern*. Ein langfristiges Forschungsziel in der Nanooptik ist die Herstellung von photonischen Bauelementen, die in ihrer Funktion der von heutigen, elektronischen Bauelementen entsprechen, jedoch eine Reihe von Vorteilen aufweisen. Solche Vorteile sind beispielsweise:

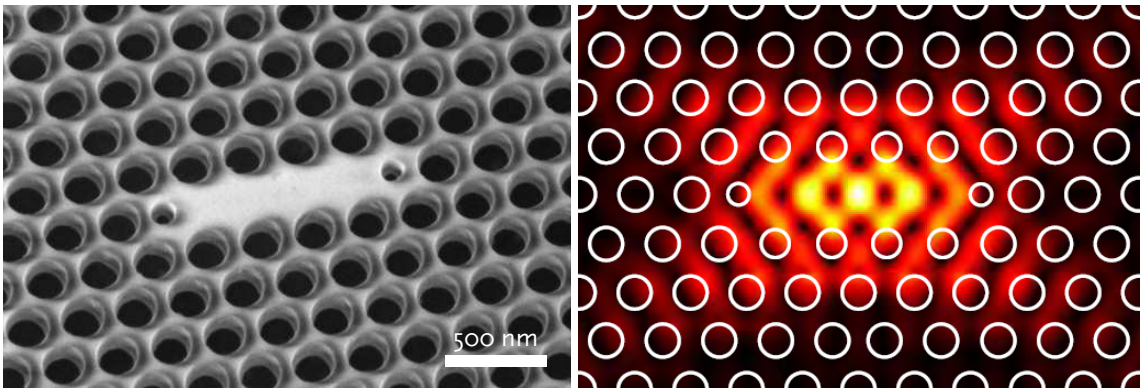
- eine nahezu verlustfreie Übertragung über weite Strecken,
- eine hohe Übertragungsgeschwindigkeit (etwa die zehnfache Geschwindigkeit elektronischer Verbindungen) und
- eine hohe Bandbreite.

Modellierung

In den Experimenten dieser Fallstudie wird auf einen photonischen Kristall ein elektromagnetischer Impuls abgegeben und dessen Ausbreitung beobachtet (siehe Abbildung 7.3b auf der nächsten Seite). Die Eigenschaften des photonischen Kristalls (z. B. Material und Struktur) und des elektromagnetischen Impulses (z. B. Stärke und räumliche Anordnung im Bezug zum Kristall) können hierbei als System aufgefasst werden, das zu untersuchen ist. Dies geschieht

[10] Die Kooperationspartner in der Nano-Fallstudie arbeiten am Institut für Physik der Humboldt-Universität zu Berlin im Bereich Nanooptik. Dieser Bereich wird geleitet von Prof. Dr. Oliver Benson. Aus seiner Arbeitsgruppe trieben Janik Wolters und Dr. Michael Barth die Kooperation maßgeblich voran. In der Nano-Fallstudie waren die METRIKer Arif Wider, Martin Schmidt (SHK) und Frank Kühnlenz beteiligt.

[11] Der Name *photonischer Kristall* ist abgeleitet von analogen Beugungs- und Reflexionseffekten von Röntgenstrahlung in Kristallen. Sie wurden erstmals 1972 von Bykov beschrieben [Byk72] und kommen auch in der Natur vor, z. B. in Form von Opalen, Vogelfedern und Schmetterlingsflügeln. Photonische Kristalle, die im sichtbaren Spektrum des Lichtes arbeiten (~350 nm (blau) bis 700 nm (rot)), sind Strukturen in der Größe der halben Wellenlänge des Lichtes.



(a) Ein photonischer Kristall in der Realität

(b) Ein photonischer Kristall im Modell bei Gabe eines elektromagnetischen Impulses

Abbildung 7.3.: Photonische Kristalle mit ~200 nm Strukturgröße in der Realität und im Modellexperiment (Bilder entnommen aus [BKS+07])

zunächst innerhalb eines geeignet gebildeten Modells und nachfolgend, bei entsprechenden, positiven Resultaten, wird das System im Labor real nachgebildet.

In den Experimenten der Fallstudie besteht der photonische Kristall aus einer dünnen Membran mit gitterförmig angeordneten Löchern. An ausgewählten Stellen werden diese Löcher ausgespart, und es bleibt ein zusammenhängender Bereich, der *Aussparung (cavity)* genannt wird. Die hauptsächliche Fragestellung für die Experimente ist es, ob eine Resonanz an diesen Aussparungen beobachtet werden kann, was allgemein das gewünschte Verhalten der Struktur bei Applizieren eines elektromagnetischen Impulses darstellt.

7.2.2. Traditioneller Experimentier-Prozess

Zur Simulation der Ausbreitung eines elektromagnetischen Impulses auf einem photonischen Kristall existieren verschiedene Simulationsverfahren in unterschiedlichen Implementierungen, die sich in Berechnungsgenauigkeit und Ressourcen-Bedarf unterscheiden. In dieser Fallstudie wurde die *Finite-Differenzen-Methode (FDM)*[12] (englisch *finite-difference time-domain (FDTD)*) angewendet [TH00, Yee66]. Die FDM ist durch zwei verschiedene Implementierungen umgesetzt, die in dieser Fallstudie von Bedeutung sind: Durch das von der Firma *Lumerical Inc.* entwickelte, kommerzielle Werkzeug *FDTD Solutions* [LS11] und durch das frei verfügbare Werkzeug *Meep* [ORI+10].

[12] Die Finite-Differenzen-Methode [Yee66] hat sich seit etwa 1990 zu der primären Modellierungstechnik in Anwendungsbereichen entwickelte, in denen Wechselwirkungen zwischen elektromagnetischen Wellen und Materialstrukturen betrachtet werden. Zu den Anwendungsbereichen zählen neben Impulsen sichtbaren Lichtes auf photonische Kristalle beispielsweise auch ultraniedrig-Frequenzbereiche in der Geophysik (1 mHz bis 10 Hz), mit denen z. B. die Erdionosphäre untersucht wird und Mikrowellen (1 bis 350 GHz), die auch zur drahtlosen Kommunikation eingesetzt werden.

In beiden Werkzeugen besteht die Beschreibung einer Experimentserie typischerweise aus folgenden vier Teilen:

1. einer Strukturbeschreibung des photonischen Kristalls, bestehend aus Material und Geometrie (Gitter, Aussparung),
2. Auflösungen in Zeit und Raum (z. B. Zahl der Gitterpunkte für die FDM),
3. der Definition einer oder mehrerer elektromagnetischer Impulsquellen und
4. Monitoren, die definieren, welche Informationen während des Simulationslaufes beobachtet werden sollen (z. B. eine zweidimensionale Fläche in einem dreidimensionalen Raum).

Die Punkte eins bis drei beschreiben das zu untersuchende System und sind neben Punkt vier Bestandteile des MUX. Das MUX ist hierbei ein Strukturmodell, das keine verhaltensbeschreibenden, mathematischen Formeln enthält. Das Verhalten der dort beschriebenen Strukturen (z. B. photonischer Kristall und elektromagnetischer Impuls) wird durch die Semantik von Schlüsselworten bestimmt, die Funktionen des interpretierenden Werkzeuges aufrufen (vor allem FDTD Solutions). Struktur- und Verhaltensteil bilden ein dynamisches, zeitdiskretes Modell.

Abbildung 7.4 auf Seite 97 visualisiert den traditionellen Experimentier-Prozess mit Sicht auf die zentralen Artefakte, wie er von den an der Fallstudie beteiligten Experimentatoren ausgeführt wird. Der Experimentator beschreibt in der textuellen Syntax von FDTD Solutions sowohl die optische Nanostruktur als auch die Parameter für das Modellexperiment (dargestellt im Anhang in Listing C.6 auf Seite 191). Er verwendet dabei einen Standard-Texteditor, der keine spezielle Unterstützung für FDTD Solutions bietet (wie z. B. Syntax-Hervorhebung oder Code-Vervollständigung). Mittels der FDTD Solutions-GUI wird aus diesem textuellen FDTD Solutions-Skript manuell eine binäre Repräsentation erzeugt bzw. kompiliert (eine Automatisierung per Kommandozeilenaufruf wird von FDTD Solutions nicht unterstützt), welche mittels FDTD Solutions ausgeführt wird. Die Auswertung der Modellergebnisse wird ebenfalls in der Syntax von FDTD Solutions textuell beschrieben. In dieser Fallstudie wird die Auswertung nicht betrachtet.

7.2.3. Probleme und Anforderungen

Obwohl die Experimentbeschreibung in beiden Werkzeugen (FDTD Solutions und Meep) konzeptionell aus den genannten vier Bestandtteilen zusammengesetzt ist, wird die konkrete Beschreibung unterschiedlich vorgenommen: In FDTD Solutions erfolgt sie entweder mittels einer komplexen GUI oder einer werkzeugspezifischen, imperativen Skriptsprache und in Meep mittels der funktionalen Programmiersprache *Lisp*.

Daraus ergibt sich das Problem, dass die gleiche Experimentbeschreibung semantisch äquivalent in verschiedenen Formaten vorgenommen werden muss, um die Stärken des jeweiligen Werkzeuges nutzen zu können. Dies bedeutet einen höheren Aufwand, ist fehleranfällig und behindert auch den Wissenstransfer zu anderen Forschergruppen, die potentiell wiederum andere Werkzeuge einsetzen. Zudem muss die reine Strukturbeschreibung möglicherweise ein

drittes mal vorgenommen werden, wenn der im Modellexperiment untersuchte, photonische Kristall im Labor produziert werden soll.

Um diese Probleme anzugehen, besteht der in dieser Fallstudie untersuchte Ansatz in der Bereitstellung einer werkzeugunabhängigen, domänenspezifischen Sprache zur Beschreibung von optischen Nanostrukturen. Diese Sprache wird als *Nano-DSL* bezeichnet und dient dem Experimentator als primäre Eingabesprache für seine Nanostrukturmodelle. Aus der Nano-DSL können entsprechende Eingabeformate für FDTD Solutions, Meep oder ein System zur Produktion der Nanostruktur im Labor automatisch transformiert werden, wodurch es diese Formate als Eingabesprachen ersetzt.[13] Die Semantik der Nano-DSL wird dabei durch die Transformationsvorschrift auf das entsprechende Zielwerkzeug abgebildet.

FDTD Solutions unterstützt als Variationsmöglichkeiten von Modelleingabeparameterbelegungen nur einfache Konstrukte über `for`-Schleifen. Wünschenswert wären aber komplexere, räumliche Variationsmöglichkeiten, wie z. B. das Verschieben von Löchern in der Nanostruktur oder von Monitoren über mehrere Experimente hinweg (in Form von Experimentserien).

Sowohl die Nano-DSL als auch die Skriptsprachen von FDTD Solutions und Meep bieten keine Unterstützung für eine Prozessbeschreibung der Experimente. Es fehlt sowohl an einer Beschreibungsmöglichkeit, welche Experimente inhaltlich zusammen zu einer Experimentserie gehören, als auch innerhalb eines Experimentes an einer Beschreibungsmöglichkeit, welche Artefakte beteiligt sind (z. B. MUX und Auswertungsskript) und wo diese eindeutig referenziert zu finden sind. Durch den Nano-DSL-Ansatz lassen sich auf der Basis einer Strukturbeschreibung verschiedene Simulationswerkzeuge einsetzen, allerdings wird es dadurch umso wichtiger, Zusammenhänge zwischen Experimenten (formal) zu beschreiben, um sie zu automatisieren und zu dokumentieren. Deshalb wurde begonnen, die Nano-DSL mit ExpL zu koppeln.

7.3. Seismologie: Erdbebenfrühwarnsystem SOSEWIN

Das Akronym SOSEWIN steht für *Self-Organizing Seismic Early Warning Information Network* und bezeichnet ein neuartiges, drahtloses, vermaschtes und dezentrales Netzwerk zur Erdbebenfrühwarnung [FPM+09]. SOSEWIN wurde im Rahmen des EU-Forschungsprojektes SAFER[14] entwickelt. Im BMBF-geförderten Projekt EDIM[15] wurde SOSEWIN in der türkischen Marmara-Region mit einer Testinstallation von 20 Knoten erprobt. Die SOSEWIN-

[13] In der aktuellen Implementierung wird lediglich eine Transformation von der Nano-DSL zu FDTD Solutions angeboten.

[14] SAFER bedeutet *Seismic eArly warning For EuRope* und bezeichnet ein Forschungsprojekt der *Europäischen Union EU* unter Beteiligung von 23 internationalen Partnern [20006b]. Die Projektlaufzeit betrug 36 Monate (ab Juni 2006). Die Entwicklung von SOSEWIN fand statt im Rahmen von *work package 4*, an dem die Projektpartner *Helmholtz-Zentrum Potsdam, Deutsches GeoForschungsZentrum (GFZ)* und Humboldt-Universität zu Berlin maßgeblich beteiligt waren.

[15] EDIM steht für *Earthquake Disaster Information System for the Marmara Region, Turkey*. Das Projekt wurde koordiniert von der Universität Karlsruhe. Neben den Projektpartnern GFZ und Humboldt-Universität zu Berlin waren maßgeblich die türkischen Partner vom KOERI beteiligt.

Fallstudie ist somit die umfangreichste der dargestellten drei Fallstudien aufgrund ihrer Einbettung in die genannten zwei internationalen Forschungsprojekte.

Die engen zeitlichen Projektvorgaben machten es erforderlich, eine pragmatische und schnell einsatzfähige Lösung für das Management von Experimenten mit SOSEWIN zu finden. ExpL als generischer Ansatz existierte zu diesem Zeitpunkt nicht und hätte mehr Forschungs- und Entwicklungsaufwand bedeutet, als im Rahmen des Projektes vertretbar gewesen wäre. Deshalb entstand die von mir spezifisch für die Projektbedürfnisse entwickelte Lösung *Experiment-Management-System (*EMS*)*.

Die SOSEWIN-Fallstudie liefert folgende Beiträge:

- Anforderungen eines forschungsrelevanten, interdisziplinären Experimentier-Prozesses für eine spezifische, MDE-basierte Entwicklung,
- eine Anwendung der Sprache ExpL für die Beschreibung von Experimenten, die in verschiedenen Ausführungsumgebungen eingesetzt werden kann (Ausführungsumgebung Prototypnetzwerk „C++-Netzwerkbibliotheken und Hardware-Treiber“ und Ausführungsumgebung „Simulationskern des Netzwerksimulators ns-3“) und
- einen Vergleich der eigenentwickelten, spezifischen Lösung EMS [FKAE09b] und dem generischen ExpL-Workflow-System.

7.3.1. Gegenstand der Untersuchung

Eine Hauptaufgabe von SOSEWIN ist die Erdbebenfrühwarnung. Zu diesem Zweck wurde ein Alarmierungsprotokoll entwickelt und plattformunabhängig modelliert. Einem MDE-Ansatz folgend, werden daraus verschiedene, plattformspezifische Modelle generiert: Für Simulationen (Einzelknoten, Kooperation von Knoten und auf Netzwerkebene), zur Virtualisierung und zur Installation auf dem realen Netzwerk.

Erdbebenfrühwarnung

Die Erdbebenfrühwarnung basiert auf dem untergrundabhängigen Geschwindigkeitsunterschied zwischen schnellen, harmlosen Primär-Wellen (*P-Wellen*) und langsamen, zerstörerischen Sekundär-Wellen (*S-Wellen*).[16] Abhängig von der Entfernung des gefährdeten Gebietes zum Hypozentrum (Ursprungspunkt des Erdbebens unter der Erdoberfläche) bleibt unterschiedlich viel Zeit für eine Erdbebenwarnung. In der Marmara-Region ergibt sich rechnerisch für die Stadt Istanbul eine maximale Frühwarnzeit von ca. 8 s. Für Mexiko-Stadt sind es beispielsweise ca. 1 min (in der Liste der weltweit am meisten bedrohten Orte stellt dies einen hohen Wert dar).

[16] Abhängig von der Geologie der spezifischen Region und des Hypozentrums bewegen sich P-Wellen mit einer Geschwindigkeit von 5-8 km/s und S-Wellen mit 3-7 km/s. Neben diesen Raumwellen unterscheidet man grundsätzlich noch Oberflächenwellen, die aber für die Erdbebenfrühwarnung keine Rolle spielen.

Erdbebenfrühwarnsysteme

Erdbebenfrühwarnsysteme, auf englisch *Earthquake Early Warning Systems (EEWSs)*, basieren auf den dargestellten Prinzipien der Erdbebenfrühwarnung. Sie haben zum Ziel, die Frühwarnzeit zu maximieren und die Anzahl von Falsch- und Fehlalarmen (*false positives* und *false negatives*) zu minimieren. EEWS bestehen aus georäumlich verteilten Knoten, die seismische Sensoren (üblicherweise Beschleunigungsmesser) für drei Achsen besitzen (zwei horizontal, eine vertikal). Von dieser Sensorik wird die P-Welle aufgezeichnet. In *zentralen EEWS* meldet jeder Knoten diese seismischen Daten oder die bereits durch ihn detektierte P-Welle an ein Datenzentrum, das über eine Erdbebenfrühwarnmeldung entscheidet (oftmals anhand eines Schwellwertes). Zentrale EEWS sind gegenwärtig der Stand der Technik [EFO+03, WCI+07, LST96].[17] SOSEWIN setzt einen neuen, *dezentralen* Ansatz zur Erdbebenfrühwarnung um, indem es ein vermaschtes, drahtloses multi-hop Netzwerk von Knoten mit seismischen Sensoren bildet.

Alarmierungsprotokoll in SOSEWIN

Das Alarmierungsprotokoll spezifiziert, wie das Netzwerk dezentral zu einer kooperativen Entscheidung über das potentiell stattfindende Erdbeben gelangt. Das Struktur und Verhalten beschreibende Modell dieses Alarmierungsprotokolls ist das MUX in dieser Fallstudie und wird im Folgenden im Überblick dargestellt (für eine ausführliche Beschreibung wird auf [FKE+08a, S. 16 ff.] verwiesen). Das MUX ist ein dynamisches, zeitdiskretes Modell, welches dem prozessorientierten Paradigma folgend, erstellt wurde.

Jeder SOSEWIN-Knoten verfügt über seismische Sensoren, um Beschleunigungswerte aufzuzeichnen, auf deren Basis er eine Signalerkennung auf mögliche P-Wellen durchführt. Das Alarmierungsprotokoll ist hierarchisch aufgebaut, um Datenrate und -volumen im Erdbebenfall gering und lokal zu halten, so dass durch *flooding* auftretende Probleme reduziert werden. Ausgezeichnete Knoten übernehmen die Rolle von Führungsknoten in einer Gruppe (*clustering*). Detektiert die Mehrheit von Knoten einer Gruppe eine P-Welle innerhalb einer bestimmten Zeit, führt dies zu einem Gruppenalarm. Wird ein Schwellwert von Gruppenalarmen im gesamten Netzwerk überschritten, kommt es zur Auslösung eines Systemalarms und einer Frühwarnmeldung.

Struktur, Verhalten und Umgebung des Alarmierungsprotokolls wurde in entsprechenden Modellen mittels einer Kombination verschiedener Sprachen spezifiziert (SDL-RT, UML, C++, ASN.1) unter Verwendung des *Real Time Developer Studio* von *PragmaDev* [AEF+09]. Auf Basis dieser Modelle sorgen entsprechende Transcompiler und Code-Generatoren dafür, dass aus denselben Modellen Code für verschiedene Ausführungsumgebungen generiert wird (MDE-Ansatz). Abbildung 7.5 visualisiert die zugrundeliegende Idee, bei der unterschiedliche Ausführungsumgebungen die Beantwortung unterschiedlicher Fragestellungen erlauben.

[17] Das *Taiwan Rapid Earthquake Information Release System (RTD)* hat für das 1999 aufgetretene Chi-Chi Erdbeben in Taiwan nach 102 s eine Frühwarnung initiiert, die bereits eine sehr gute Näherung für das Hypozentrum und die Magnitude beinhaltete [WLC+00].

Die jeweils gewonnenen Resultate führen zur Verbesserung derselben Modelle: Simulativ erkannte Probleme können im Modell behoben werden und mittels Code-Generierung wird auch die auf den Knoten installierte Software entsprechend verbessert.

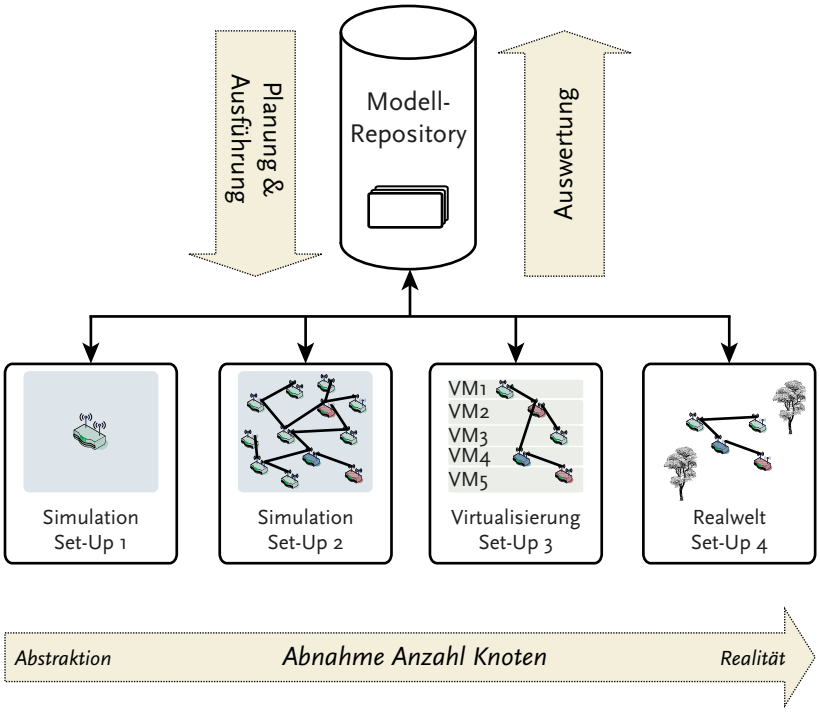


Abbildung 7.5.: Modellbasierter Ansatz zur Entwicklung des Alarmierungsprotokolls in SOSEWIN mit verschiedenen Ausführungsumgebungen

Simulation mittels ODEMx Zur Ermittlung regionenspezifischer Parameter des P-Wellendetektionsalgorithmusses wurde Set-Up 1 in Abbildung 7.5 verwendet. Dabei wird ein einzelner Knoten mit Hilfe der Simulationsbibliothek ODEMx [FAWG07, Ger03, KFA07] modelliert, so dass er mit real aufgezeichneten oder synthetischen Erdbebenwellendaten versorgt werden kann. Set-Up 2a in Abbildung 7.5 beinhaltet Set-Up 1 und wurde um ein stark vereinfachtes ODEMx-basiertes Netzwerkmodell erweitert. Mit Hilfe von Set-Up 2a lassen sich Aussagen zu Anzahl und Größe von Nachrichten auf Applikationsebene treffen. Der Vorteil dieses ODEMx-basierten Simulators ist die Betrachtung von potentiell großen Netzwerktopologien.[18] Allerdings sind bestimmte Fragestellungen, wie z. B. die Laufzeit von Nachrichten im SOSEWIN, mit diesem Ansatz nicht zu beantworten: Durch eine intendierte Abstraktion wird nur konstanter Zeitverbrauch pro Hop angenommen. Durch den Einsatz von Mittelwerten aus Beobachtungen am realen System kann man diese Zeit abschätzen; doch die Aussagekraft ist begrenzt, weshalb weitere Methoden zum Einsatz kommen.

[18] Bisherige Untersuchungen konnten dabei harte Grenzen im Ressourcenbedarf (z. B. CPU-Zeit, Speicher) nicht aufzeigen, so dass erwartet werden kann, damit tausende von kommunizierenden Knoten untersuchen zu können (es wurden SOSEWINs mit eintausend Knoten simuliert).

Virtualisierung Eine weitere Methode, bezeichnet als Set-Up 3 in Abbildung 7.5, ist der Einsatz von Virtulisierung, um das Alarmierungsprotokoll hinsichtlich Schnittstellen zu anderen Softwarekomponenten (Betriebssystem, Netzwerk-Interfaces, usw.) in einer identischen Umgebung wie auf den realen Knoten ablaufen zu lassen. Die Gesamtheit dieser Komponenten wird als *Virtuelle Maschine (VM)* verstanden. Zweck der Virtualisierung ist ein Integrationstest aller Softwarekomponenten, wie sie auch im realen System zu finden sind. Von Vorteil ist dabei, nicht auf das Vorhandensein von Knoten-Hardware angewiesen zu sein und trotzdem das Alarmierungsprotokoll unter Bedingungen zu betreiben, die näher an der Realität sind, als in einem Simulationskontext. Allerdings benötigt eine solche Virtualisierung sehr viel mehr Ressourcen, als eine Simulation. Daher ist die Anzahl der Knoten stark eingeschränkt (ca. 30 auf dem eingesetzten Virtualisierungsserver).

Netzwerksimulation mittels ns-3 Um Aussagen über Frühwarnzeiten unter einem realistischeren Netzwerkmodell zu treffen, als das bei der ODEMx-basierten Simulation verwendete, wurde ein Netzwerksimulator eingesetzt (Set-Up 2b in Abbildung 7.5 auf der vorangegangenen Seite). Hierfür wurde das Alarmierungsprotokoll für den *network simulator 3 (ns-3)* [HRFR06] neu modelliert.[19] Ein Netzwerksimulator wie ns-3 enthält typischerweise Modelle, welche die physikalische Ausbreitung von Funkwellen abbilden, Implementierungen verschiedener WLAN-Standards und Routing-Protokolle. Damit sind realistischere Aussagen bezüglich Laufzeiten von Nachrichten möglich, die u. a. durch die genommenen Wege (Routen) und Kollisionen beeinflusst werden. Der Aufwand für Simulationen dieser Art liegt zwischen der beschriebenen ODEMx-basierten Simulation und der Virtualisierung, so dass Netze mit einer Knotenanzahl von mehreren hundert untersucht werden können.

7.3.2. Traditioneller Experimentier-Prozess

Für jedes Experiment auf Basis des Alarmierungsprotokolls in den beschriebenen Ausführungsumgebungen muss eine Netzwerktopologie (georäumliche Anordnung der Knoten, bidirektionale Links und Gruppenstruktur) angegeben werden. Zudem muss jeder Knoten seismische Sensordaten zur Verfügung gestellt bekommen. Sowohl die Netzwerktopologie als auch die Sensordaten können entweder aus der Realwelt aufgezeichnet sein oder synthetisch erzeugt werden. Abbildung 7.6 auf Seite 98 visualisiert den Teilprozess Datenaquise, durch welchen eine Netzwerktopologie und Sensordaten für ein Experiment bereitgestellt werden.

Dabei ergeben sich drei mögliche Szenarien:

1. Ein Erdbeben aus dem historischen Erdbebenkatalog wird ausgewählt (Netzwerktopologie und Sensordaten sind fix),
2. eine historische Netzwerktopologie wird mit synthetischen Sensordaten versorgt, und
3. eine synthetische Netzwerktopologie wird mit synthetischen Sensordaten versorgt.

[19] Die Arbeit von Brumbulli et al. beschäftigt sich damit, eine automatische Transformation von Modellen der *Specification- and Description-Language (SDL)* nach ns-3 zu ermöglichen [BF11, BBEF11].

Abbildung 7.7 auf Seite 99 visualisiert sowohl vorbereitende Aktivitäten als auch den gesamten Experimentier-Prozess in SOSEWIN. Zu den vorbereitenden Aktivitäten (im graphisch ausgezeichneten Bereich Vorbereitungsaktivitäten) zählen die Katalogisierung von historischen Erdbeben und existierenden Netzwerktopologien sowie die Modellierung von Alarmierungsprotokoll, Erdbebenwellendaten (zur Erzeugung von synthetischen Sensordaten pro Knoten nach der Methode von Wang [Wan99]) und synthetischer Netzwerktopologien[20].

Abbildung 7.7 auf Seite 99 beeinhaltet im graphisch ausgezeichneten Bereich Modellierung und Ausführung von Experimenten den Teilprozess Datenaquise aus Abbildung 7.6 auf Seite 98, um die notwendigen Eingabedaten für das Modell des Alarmierungsprotokolls bereit zu stellen. Die Aktivität Transformieren ruft den eigenentwickelten HUB-Transcompiler [FKAE09b] auf, der aus dem plattformunabhängigen Modell des Alarmierungsprotokolls ein plattformspezifisches generiert, das innerhalb der geplanten Ausführungsumgebung ablaufen soll.

Zur Vereinfachung von Abbildung 7.7 wurde das Parametrisieren in Experimentserien nicht dargestellt (z. B. im Teilprozess Datenaquise, innerhalb der Aktivität Transformieren oder für die Ausführung der plattformspezifischen Modelle). Zudem wurde die Ausführung der plattformspezifischen Modelle in Set-Up eins bis vier als parallele Aktivitäten gezeigt, was eine Optimierung im Vergleich zu einem möglichen sequentiellen Ablauf darstellt.

Die Modellbeobachtungen aus den Abläufen in allen Ausführungsumgebungen werden gesammelt (durch Log-Dateien) und im Experiment-Repositorium einheitlich gespeichert. Die Konsistenz hinsichtlich aller Aktivitäten, die zu den Eingabedaten geführt haben, ist dabei gesichert. Das einheitliche Format der Modellbeobachtungen verringert signifikant den Aufwand in der Auswertung des Datenmaterials, insbesondere den Vergleich zwischen Experimenten in verschiedenen Ausführungsumgebungen.

Experiment-Management mittels EMS

Der in Abbildung 7.7 auf Seite 99 dargestellte Experimentier-Prozess wird durch ein Experimentier-Computersystem unterstützt, das aus den Werzeugen *Experiment-Management-System (*EMS*)* [FKAE09b] und *uDig Erdbeben-Edition (uDigEE)* [Eve08] besteht. Man kann EMS als Workflow-Management-System verstehen, bei dem Typ, Reihenfolge und Parameter der Aktivitäten durch den Quellcode fest vorgegeben sind. Eine solche Workflow-Definition wird im Folgenden als *parametrisierte, feste Workflow-Definition* bezeichnet. Die Festlegung der Parameterwerte für die Aktivitäten geschieht mit Hilfe der graphischen Oberfläche von EMS oder über uDigEE, falls es sich um georäumliche Parameter handelt.[21]

Abbildung 7.8 auf Seite 100 zeigt die Werkzeuge und Artefakte im Experimentier-Prozess in SOSEWIN, wobei zur besseren Orientierung die drei Phasen eines Experimentier-Workflows

[20] Zur Erzeugung synthetischer Netzwerktopologien wird ein Grid-basierter Ansatz verwendet, bei dem eine spezifizierte Anzahl an Knoten georäumlich äquidistant innerhalb eines festgelegten Gebietes automatisch angeordnet wird. Die Verwendung von realistischeren georäumlichen Anordnungen wäre jedoch wünschenswert.

[21] Eine Parameterwertebelegung der parametrisierten, festen Workflow-Definition entspricht der Definition eines Experimentes aus Abschnitt 2.1.1 auf Seite 19.

farblich unterlegt sind. In der Planungsphase (oberes Kreissegment in Abbildung 7.8) werden Artefakte (z. B. das Alarmierungsprotokoll-Modell), deren Konfiguration (z. B. Variation von Modelleingabeparameterbelegungen) und die Ausführungsumgebung ausgewählt. In der Ausführungsphase (rechtes Kreissegment in Abbildung 7.8) wird automatisch zu einem konfigurierten Zeitpunkt das gewünschte Experiment vorbereitet (z. B. synthetische Sensordaten erzeugt), ausgeführt, überwacht und dokumentiert. Die Ergebnisse der Modellausführung werden aus den in einem Zwischenschritt produzierten Log-Dateien in das Experiment-Repositorium importiert (umgesetzt durch eine relationale Datenbank mit GIS-Funktionalität). Dadurch wird eine einheitliche Sicht auf die Experimentergebnisse angeboten, die von der Ausführungsumgebung der zugrundeliegenden Experimente abstrahiert und somit die Erstellung von Auswertungsverfahren vereinfacht. Das Experiment-Repositorium enthält zudem die gesamte Konfiguration jedes durchgeführten Experimentes. Somit sind die Konsistenz und Vollständigkeit der Experiment-Artefakte sichergestellt. Das ist eine Bedingung für Reproduzierbarkeit bzw. Wiederholbarkeit der Experimente.

In der Auswertungsphase (linkes Kreissegment in Abbildung 7.8 auf Seite 100) ist auf Basis der Daten im Experiment-Repositorium eine (semi-)manuelle Auswertung möglich. Zur Auswertung von Experimenten (Auswertungsphase, linkes Kreissegment in Abbildung 7.8 auf Seite 100) bietet EMS eine Menge von Verfahren an, wobei jedes Verfahren wiederum als feste, parametrisierte (Sub-)Workflow-Definition verstanden werden kann, die als EMS-*Auswertungsworkflow* bezeichnet wird. Es existieren EMS-Auswertungsworkflows für einzelne Experimente oder für Experimentserien.[22] EMS-Auswertungsworkflows erzeugen z. B. Diagramme, die bestimmte zeitliche Kenngrößen (z. B. erste P-Wellendetektion im Netzwerk oder Systemalarm) visualisieren oder auch die Anzahl und die Größe von übertragenen Nachrichten.

7.3.3. Probleme und Anforderungen

Der Experimentier-Prozess mit dem SOSEWIN-Alarmierungsprotokoll umfasst Experimentserien, die in drei Gruppen von Variationen unterteilt werden können:[23]

Detektionsalgorithmus Zur Erkennung einer P-Welle verwendet ein Knoten einen bestimmten Detektionsalgorithmus. Sowohl der Detektionsalgorithmus selbst als auch dessen Parameterbelegung müssen variiert werden können.

Hypozentrum Das Hypozentrum bezeichnet den Ursprung des Erbebens und wird durch eine georäumliche Koordinate repräsentiert. Die Position des Hypozentrums im Bezug zur Entfernung zum SOSEWIN soll variiert werden, um dessen Frühwarngüte beurteilen zu können. Dabei sind Experimentserien durchzuführen, in denen das Hypozentrum in äquidistanten Schritten auf einer gedachten Gerade näher an das SOSEWIN herankommt.

[22] EMS-Auswertungsworkflows werden manuell gestartet; das automatische Ausführen im Anschluss an ein Experiment oder eine Experimentserie ist in EMS nicht möglich (aber wünschenswert).

[23] Für eine detaillierte Darstellung dieses Parameterraumes wird auf [FKE+09d] verwiesen.

Erdbebenstärke Die Erdbebenstärke (Magnitude) als Einflussfaktor auf die Erkennungsgüte steht im Zusammenhang mit der Distanz des Hypozentrums zum SOSEWIN. Bei gleicher Distanz können theoretisch stärkere Beben zuverlässiger erkannt werden als schwächere. Bei diesen Experimentserien wird die Magnitude als skalarer Wert verändert (aus dem Bereich der reellen Zahlen).

Netzwerktopologie Die Variation der Netzwerktopologie umfasst die Variation der georäumlichen Positionen der Knoten, deren Konnektivität (z. B. symmetrische oder asymmetrische Links, Übertragungswahrscheinlichkeiten), die Cluster- bzw. Gruppenbildung und die Position von Gateway-Knoten (zuständig für die Kommunikation nach außerhalb des SOSEWIN, um beispielsweise eine Erdbebenalarmmeldung abzusetzen).

Die Probleme beim Experimentieren mit dem SOSEWIN-Alarmierungsprotokoll liegen in den Phasen Planung und Auswertung. Die Ausführung der Experimente wird durch die Werkzeuge EMS und uDigEE zufriedenstellend gelöst, da die Aktivitäten (und ihre Reihenfolge) dem in Abschnitt 7.3.2 auf Seite 84 beschriebenen, festen Ablauf folgen.

Probleme in der Planungsphase

Das traditionelle Vorgehen zur Erstellung neuer Experimente ist, mittels EMS eine vorhandene Parameterbelegung eines Experimentes zu kopieren und anzupassen, so dass dadurch das gewünschte, neue Experiment erstellt wird. Die Variation von Parameterbelegungen lässt sich auf diese Weise manuell betreiben, doch dies ist zeitaufwändig und fehleranfällig, so dass sich damit die Untersuchung von großen Parameterräumen durch die resultierende Vielzahl an Experimenten nur unzureichend planen lässt.

Die Variation der Position von Punkten in einem georäumlichen Koordinatensystem ist eine für mehrere Experimentserien (Hypozentrum und Netzwerktopologie) notwendige Anforderung, die aber nur unzureichend durch die Werkzeuge EMS und uDigEE erfüllt wird. Die Position des Hypozentrums für ein Experiment wird beispielsweise manuell in der GIS-basierten Oberfläche von uDigEE geändert. Gleiches gilt für die Netzwerktopologie, für deren Erzeugung uDigEE Assistenz bietet, um Grid-basierte Topologien zu erzeugen. Hierzu wird ein rechteckiger Bereich markiert und die Anzahl der zu verteilenden Knoten eingegeben, woraus dann eine äquidistante Anordnung der gewünschten Knoten innerhalb des markierten Gebietes vorgenommen wird. Es mangelt somit an einer geeigneten Beschreibung von Variationen in Experimentserien, insbesondere bei Variationen georäumlicher Parameterbelegungen.

Probleme in der Auswertungsphase

Alle EMS-Auswertungsworkflows sind Teil einer semi-automatischen Auswertungsphase (Abschnitt 4.3.3.2). Sie produzieren Daten, die durch den Experimentator ausgewertet werden müssen. Es fehlt an Auswertungsworkflows, deren Ergebnisse durch EMS automatisch ausgewertet werden können (Vollautomatische Auswertungsphase). Hierzu wurde das Konzept einer Bewertungsfunktion vorgestellt (Abschnitt 4.3.3.3), auf deren Basis eine qualitative Vergleichbarkeit von Experimentergebnissen möglich wäre.

Für die vorgestellten Gruppen von Experimentserien lassen sich beispielsweise folgende Bewertungsansätze angeben, die als Bewertungsfunktion in Auswertungsworkflows umgesetzt werden könnten:

Detektionsalgorithmus Wieviele Knoten haben autark eine P-Welle detektiert? Wie hoch sind die Anzahlen an falsch positiven und falsch negativen Detektionen pro Knoten? Geringe Werte stellen eine hohe Erkennungsgüte dar.

Hypozentrum, Erdbebenstärke, Netzwerktopologie Es wurde projektspezifisch definiert, dass eine Frühwarnung erfolgreich möglich ist, wenn gilt: $t_{det} + t_{reakt} < t_{ankunft}$. Dabei ist t_{det} die Zeit für die Erdbebenerkennung in SOSEWIN ab dem Eintreffen der P-Welle an der Position des Netzwerkes und $t_{ankunft}$ die Zeit bis zum Eintreffen der S-Welle am zu schützenden Ort. Die Reaktionszeit t_{reakt} gibt den Zeitbedarf für die Durchführung von akuten Katastrophenschutzmaßnahmen nach Eintreffen des Systemalarmes an. Die Reaktionszeit liegt damit außerhalb des Einflussbereiches des SOSEWIN und kann als konstant angenommen werden. Entsprechend ist das Ziel, t_{det} zu minimieren. Anhand des Wertes von t_{det} kann man eine qualitative Bewertung von Experimentergebnissen durchführen.

Die Eingabeparemeter Hypozentrum und Erdbebenstärke sind Invarianten, so dass die Güte der Frühwarnung nur durch Variationen in der Netzwerktopologie, dem Alarmierungsprotokoll und dem Detektionsalgorithmus verbessert werden kann. Das Ziel, t_{det} zu minimieren, soll gleichzeitig mit dem Minimieren von falsch positiven und falsch negativen Systemalarmen erreicht werden. Dadurch ergibt sich ein Polyoptimierungsproblem.

Rückkopplung zwischen Experimenten Um solche Polyoptimierungsprobleme werkzeugunterstützt zu lösen, ist eine Rückkopplung zwischen Experimentergebnissen und der Planung neuer Experimentserien notwendig. So können beispielsweise die in einer Experimentserie ermittelten, hinsichtlich der Erkennungsgüte optimalen Parameterwerte des Detektionsalgorithmusses in weiteren Experimentserien als invariant betrachtet werden.

Einbeziehung von Daten der Realwelt-Ausführungsumgebung Das Experimentieren mit dem SOSEWIN-Alarmierungsprotokoll findet auch in einer Realwelt-Ausführungsumgebung statt (Abschnitt 7.3.1 auf Seite 82 und Set-Up 3 in Abbildung 7.5). Dabei treten typischerweise durch die reale Umgebung verursachte Phänome auf, die unerwartet sind und deren Vorkommen erkannt und in der Wechselwirkung auf das Experiment analysiert werden muss. So wird beispielsweise die Linkqualität der Verbindung zwischen zwei Knoten durch Objekte in der Umwelt (z. B. Bäume) stark beeinflusst. Die Linkqualität beeinflusst, ob und in welcher Verzögerung Nachrichten des Alarmierungsprotokolles zugestellt werden. Das Überwachen der Linkqualitäten kann somit wertvolle Hinweise geben, warum eine Frühwarnung im Experiment möglicherweise nicht erfolgreich gewesen ist. Monitoring-Daten, wie z. B. die Linkqualitäten, müssen somit während des Experimentes aufgezeichnet werden können, ohne das Experiment zu beeinflussen, und in Beziehung zum Ablauf des Experimentes gesetzt werden können.

7.4. Anforderungen an die Beschreibung von Experimentier-Prozessen

Anhand der Methodik von Modellierung & Simulation (siehe Abschnitt 2.2) und den durchgeführten Fallstudien konnte ein Katalog von gemeinsamen, generellen Anforderungen an die Beschreibung von Experimentier-Prozessen mit computerbasierten Modellen identifiziert werden, der in diesem Abschnitt 7.4 dargestellt ist.

Die Anforderungen an die Beschreibung von Experimentier-Prozessen lassen sich einteilen in *strukturell*, *semantisch* oder *strukturell und semantisch* (in Abbildung 7.9 gekennzeichnet mittels Stereotypen). *Strukturell* bezeichnet hierbei, dass eine Eigenschaft sich auf die Struktur eines Ablaufes oder die Struktur von Daten bezieht. Strukturelle Eigenschaften lassen sich oftmals automatisiert überprüfen.[24] Im Gegensatz dazu lassen sich *semantische* Eigenschaften nicht automatisiert überprüfen (z. B. die Verständlichkeit).

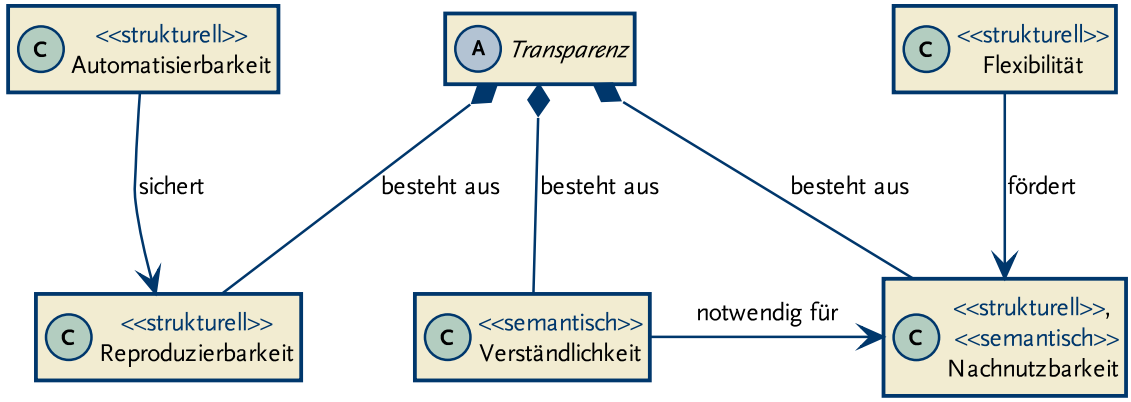


Abbildung 7.9.: Relationen zwischen Anforderungen an die Beschreibung von Experimentier-Prozessen (als UML-Klassendiagramm)

7.4.1. Reproduzierbarkeit

Ein Grundpfeiler des wissenschaftlichen Arbeitens ist es, dass Wissenschaftler ihre Methoden zur Erlangung bestimmter Resultate derart zur Verfügung stellen, dass andere Wissenschaftler diese Resultate unter gleichen Bedingen ebenfalls erzielen können.[25] „Alle Resultate eines Experimentes oder einer Studie zu dokumentieren, und die Primärdaten zu sichern und aufzubewahren“ wird entsprechend als eines der allgemeinen Prinzipien guter, wissenschaftlicher

[24] Für solche Überprüfungen werden *Traces* mehrerer Ausführungen des zu überprüfenden Workflows benötigt. Dabei müssen diese Traces den Kontrollfluß zwischen den Aktivitätsinstanzen und die Datenflüsse zwischen ihnen aufzeigen.

[25] „Nur dort, wo gewisse Vorgänge (Experimente) auf Grund von Gesetzmäßigkeiten sich wiederholen, bzw. reproduziert werden können, nur dort können Beobachtungen, die wir gemacht haben, grundsätzlich von jedermann nachgeprüft werden.“ [Pop05, S. 22] „Die *Objektivität* der wissenschaftlichen Sätze liegt darin, daß sie *intersubjektiv nachprüfbar* sein müssen.“ [Pop05, S. 21]

Praxis angesehen [Ber02]. Der Autor der hier vorliegenden Arbeit zählt zu den erwähnten Primärdaten insbesondere Experimentier-Workflows.

Reproduzierbarkeit (*reproducibility*) bezeichnet:

> „the ability for a third party who has access to the description of the original experiment and its results to reproduce those results using a possibly different setting, with the goal to to confirm or dispute the original experimenter's claims" [MWHW12].

Nach dieser Definition steht die Reproduzierbarkeit des Experimentergebnisses im Vordergrund, wobei Abweichungen im Aufbau (*setting*) des Experimentes erlaubt sind. Man definiert sich hierbei ein dem Untersuchungsziel angemessenes Ähnlichkeitsmaß und bewertet auf dessen Basis die Gleichheit von Ergebnissen.

Beim Experimentieren in realweltlichen Szenarien wirken Umwelteinflüsse von außen auf das betrachtete System ein, die nicht modelliert werden können oder vereinfachend nicht modelliert werden sollen. Misst man beispielsweise Latenzen von Nachrichten, die in einem drahtlosen Maschennetzwerk übertragen werden, so variieren diese Latenzen durch Umwelteinflüsse, wie beispielsweise durch Objekte, die sich zwischen den Knoten des Netzwerkes bewegen (z. B. Blätter, Vögel, u. ä.). Exakt gleiche Meßergebnisse innerhalb von verschiedenen Ausführungen eines Experimentes in diesem Szenario werden nicht auftreten, je nach Untersuchungsziel ergibt sich aber durchaus das selbe Experimentergebnis (z. B. nach Anwendung statistischer Verfahren auf den Meßergebnissen).

Die *Wiederholbarkeit* (*repeatability*[26]) eines Experimentier-Prozesses ist gegeben, wenn er wiederholt mit *exakt gleichem* Ergebnis ausgeführt werden kann. Wiederholbarkeit ist somit strenger als Reproduzierbarkeit gefasst. Zudem ist sie folglich nicht in allen Szenarien der Ausführung eines MUX erreichbar. Bestimmte Ausführungsumgebungen, wie die realweltliche im obigen Beispiel, schließen Wiederholbarkeit nach dieser Definition oftmals aus. Die Beschreibung des Experimentier-Prozesses muss alle für die Wiederholbarkeit notwendigen Aktivitäten in der korrekten Reihenfolge enthalten und auch alle notwendigen Eingabedaten spezifizieren. Die Wiederholbarkeit ist eine wichtige, strukturelle Eigenschaft des Experimentier-Prozesses, um der wissenschaftlichen Methodik zu genügen. Hierfür muss das MUX geeignet modelliert worden sein (wird beispielsweise Pseudo-Zufall verwendet, so muss der entsprechende Initialisierungsvektor als Modelleingabeparameter spezifiziert sein).

7.4.2. Automatisierbarkeit

Automatisierbarkeit durch ein Computersystem bedeutet, dass der zugrundeliegende Experimentier-Prozess nur aus automatisierten Aktivitäten besteht (was der Definition eines Experimentier-Workflows entspricht), die durch ein Computersystem ausgeführt werden können. Die Automatisierbarkeit stellt i. A. sicher, dass der Experimentier-Workflow reproduzierbar ist, sofern alle dafür benötigten Software-Artefakte für das Computersystem verfügbar sind.

[26] Wiederholbarkeit ist nach Missier „the attempt to replicate an experiment and obtain the same exact results when no changes occur anywhere" [MWHW12].

7.4.3. Verständlichkeit

Während Reproduzierbarkeit und Automatisierbarkeit rein strukturelle Eigenschaften sind, ist die Verständlichkeit eine rein semantische und als solche nicht quantitativ messbar. Die folgenden Fragen sollen helfen, die Verständlichkeit eines Experimentier-Prozesses einzuschätzen:

- Welches Untersuchungsziel bzw. welche Fragestellung soll mit Hilfe des Experimentier-Prozesses beantwortet werden?
- Welche konkreten Experimente werden durch den Experimentier-Prozess definiert?
- Wie stehen Experimente untereinander in Zusammenhang (bilden sie Experimentserien?)?
- Welche Aktivitäten werden in welcher Reihenfolge ausgeführt und was bewirken sie einzeln und in der Komposition?
- Wie sind die Experimentergebnisse entstanden?

Die letzte Frage der Herkunft von Experimentergebnisdaten wird unter der Bezeichnung *Provenance* zusammengefasst. Die *W3C Provenance Incubator Group* definiert Provenance in [Gro10] wie folgt:

> Provenance refers to the sources of information, such as entities and processes, involved in producing or delivering an artifact. The provenance of information is crucial to making determinations about whether information is trusted […].

Die Provenance der Experimentergebnisdaten schließt auch temporäre Daten ein, die durch Aktivitäten für eine weitere Verarbeitung zuvor erzeugt wurden. Die Kenntnis der Provenance hilft, Vertrauen zu erlangen, dass ein Workflow überhaupt korrekte Ergebnisse produzieren kann (was nicht bedeutet, dass er es tatsächlich tut).

7.4.4. Nachnutzbarkeit

Die Nachnutzbarkeit eines Experimentier-Prozesses bedeutet, dass er in Teilen oder als Ganzes erneut in einem anderen Experimentier-Prozess (eventuell abgeändert) wiederverwendet werden kann. Dies setzt auf semantischer Ebene voraus, dass er die nötige Verständlichkeit aufweist, um zu entscheiden, ob er für die neue Experimentier-Zielstellung geeignet ist. Beispielsweise müssen das MUX und die Eingabedaten geeignet sein. Ist ein Experimentier-Prozess nachnutzbar, kann er durch den Experimentator selbst oder durch die wissenschaftliche Gemeinschaft wiederverwendet werden. Der strukturelle Aspekt der Nachnutzbarkeit ist die Flexibilität.

7.4.5. Flexibilität

Flexibilität ist eine strukturelle Anforderung und bedeutet, dass die Teilbeschreibungen des Experimentier-Prozesses komponierbare Einheiten bilden sollen, die wiederverwendbar und

in ihrer Reihenfolge austauschbar sind. Bei Veränderung einer Teilbeschreibung sollen die vorangegangenen und nachfolgenden Teilbeschreibungen möglichst unverändert übernommen werden können. Dies vereinfacht die Erstellung neuer Experimentier-Prozesse und die Nachnutzbarkeit bereits vorhandener.

7.4.6. Transparenz

Transparenz ist in dieser Arbeit ein abstrakter Oberbegriff für *Verständlichkeit*, *Nachnutzbarkeit* und *Reproduzierbarkeit* und fasst die grundlegenden Anforderungen an wissenschaftliches Arbeiten auf Basis der Methodik des Experimentierens mit computerbasierten Modellen zusammen. Abhängig von der Experimentier-Domäne können sich zudem spezielle Anforderungen ergeben (z. B. wird in Experimenten mit drahtlosen Computernetzwerken die Transparenz erhöht, indem konform zu einer Empfehlung der IEEE 802.16 Arbeitsgruppe experimentiert wird). Douglass B. Lee bemerkt über Transparenz bei Modellen allgemein [Lee73]:

> Probably the most important attribute any model should have is *transparency*. It should be readily understandable to any potential user with a reasonable investment of effort.

Diese Bemerkung trifft nicht nur auf die Modelle selbst zu, sondern gilt auch für den Prozess des Experimentierens mit diesen Modellen.

7.5. Anforderungen an ein Computersystem für Experimentier-Prozesse

In diesem Abschnitt 7.5 werden Anforderungen an ein Computersystem formuliert, welches die Anforderungen an die Beschreibung von Experimentier-Prozessen erfüllen kann. Ein solch geeignetes Computersystem soll *Experimentier-Computersystem* genannt werden. Der Begriff *Experimentier-Computersystem* bezeichnet die Gesamtheit von Experimentier-Prozessbeschreibungen (mit zugehörigen Instanzen) und dem Management-System, das sie verwaltet.[27] Die Bestandteile eines Experimentier-Computersystem werden in Abbildung 7.10 visualisiert und nachfolgend erläutert.

7.5.1. Funktionale Anforderungen

Experimentier-Management-System

Reproduzierbarkeit und Automatisierbarkeit des Experimentier-Prozesses mit computerbasierten Modellen erfordern ein Experimentier-Management-System, das die Eigenschaften

[27] Der Begriff *Experimentier-Computersystem* ist damit analog zu dem des *Workflow-Systems* aus Kapitel 5.

Atomarität, Konsistenz, Isoliertheit und Dauerhaftigkeit (AKID)[28] garantiert. Zur besseren Veranschaulichung wird bei den Erläuterungen zu den AKID-Eigenschaften auf ein WfMS Bezug genommen, wie es in dieser Arbeit entwickelt wurde – grundsätzlich können jedoch auch andere Klassen von Systemen die genannten Anforderungen erfüllen. Die AKID-Eigenschaften sind:

Atomarität (Abgeschlossenheit) Eine Sequenz von atomaren Elementaraufgaben einer Aktivität wird entweder vollständig oder gar nicht ausgeführt. Entsprechend stellt eine Aktivität typischerweise die kleinste Arbeitseinheit dar, die von einem WfMS verwaltet wird.

Konsistenz Jede Sequenz von Datenoperationen (in einer Aktivität) hinterlässt nach Beendigung einen konsistenten Datenzustand, falls der Ausgangszustand davor ebenfalls konsistent war. Konsistenz bezieht sich hierbei darauf, dass Workflow-relevanten Daten und Workflow-Kontrolldaten nach Beendigung einer Aktivität wiederum der Experimentier-Workflow-Definition genügen.

Isoliertheit (Abgrenzung) Nebenläufig in Ausführung befindliche Datenoperationen dürfen sich nicht (oder nur definiert eingeschränkt) gegenseitig beeinflussen. Die gilt insbesondere für nebenläufig ausgeführte Experimente (als Aktivitäten eines Experimentier-Workflows).

Dauerhaftigkeit Der Experimentier-Workflow und alle Ergebnisdaten müssen dauerhaft gespeichert werden. Dies schließt auch alle Daten ein, die während des Ablaufes der Workflow-Instanz über die Herkunft der Ergebnisdaten entstanden sind (Monitoring-Informationen). Es muss garantiert sein, dass weder der Workflow, noch die durch seine Ausführung produzierten Daten verändert werden.

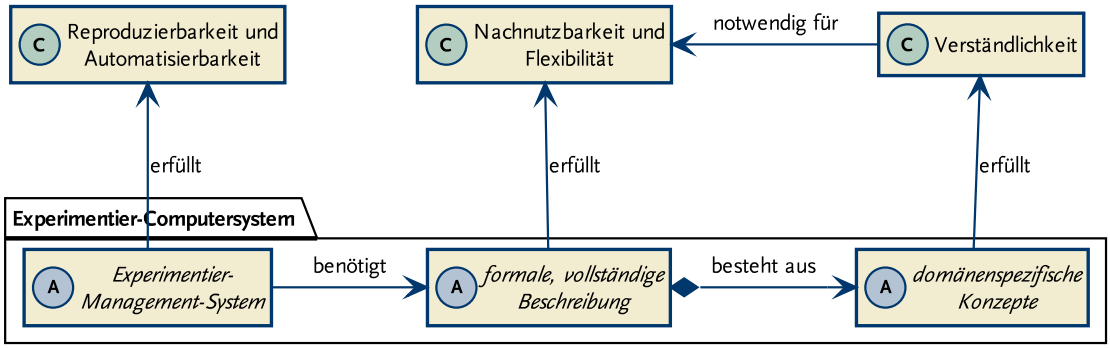


Abbildung 7.10.: Relationen zwischen Anforderungen an die Beschreibung von Experimentier-Prozessen und deren Umsetzung durch ein Computersystem (als UML-Klassendiagramm; erweitert Abbildung 7.9 auf Seite 89)

[28] AKID-Eigenschaften beschreiben erwünschte Eigenschaften von Verarbeitungsschritten in Datenbankmanagementsystemen und sind definiert über Sequenzen von Datenoperationen. In diesem Kontext werden solche Sequenzen von Datenoperationen durch Aktivitäten eines Workflows ausgeführt.

Domänenspezifische Konzepte

Jeder Experimentator besitzt (fundiertes) Wissen über die Domäne, in der er seine Experimente durchführt, so dass er darin ein Domänen-Experte ist. Domänenspezifische Konzepte des Experimentierens verwendet ein Experimentator somit bereits implizit und intuitiv zur Beschreibung seiner Experimente, ohne sie explizit zu benennen. Dazu zählen beispielsweise der Experimentbegriff, Reihenfolgen von Experimenten (sequentiell, parallel) oder auch die Variation von Modelleingabeparameterbelegungen. Definiert man diese Konzepte explizit, erhöht sich die Verständlichkeit des Prozesses nicht nur für den Experimentator, der die Beschreibung erstmalig erstellt, sondern auch für jeden in der wissenschaftlichen Gemeinschaft, der diesen Prozess verstehen bzw. als Basis für eigene Forschungsvorhaben nutzen möchte.

Formale, vollständige Beschreibung

Eine formale, vollständige Beschreibung des Experimentier-Prozesses zu erstellen, zu verwalten und kollaborativ bearbeiten zu können, sind die zentralen Anforderungen für ein Experimentier-Computersystem. Kollaboratives Arbeiten bedeutet, dass mehrere Experimentatoren an einer Beschreibung eines Experimentier-Prozesses arbeiten; dies kann gleichzeitiges Editieren, Editieren und Betrachten oder auch einen (teilweisen oder vollständigen) Export aus dem Experimentier-Computersystem bedeuten (z. B. in Form einer Visualisierung). Die Erfahrungen aus den Fallstudien haben hierbei gezeigt, dass domänenspezifische Konzepte des Experimentierens in Struktur und Semantik der Beschreibung wiederzufinden sein sollen, da sich dadurch die Verständlichkeit der damit beschriebenen Experimentier-Prozesse erhöht.

Das Attribut *formal* bedeutet, dass die Beschreibung ausschließlich aus Elementen besteht, die eine definierte Struktur bilden, welche durch Produktionsregeln bestimmt ist. Diese definierte Struktur (ihrerseits selbst eine formale Beschreibung) besteht aus typisierten Relationen (z. B. enthalten-sein, ist-ein, assoziiert) zwischen bedeutungstragenden Elementen, die anhand der domänenspezifischen Konzepte identifiziert werden. Eine in diesem Sinne formale Beschreibung erfüllt die strukturellen Anforderungen der Nachnutzbarkeit und Flexibilität.

Mit dem Attribut *vollständig* ist gemeint, dass sämtliche strukturellen und semantischen Informationen formal beschreibbar sein müssen, um einen Experimentier-Prozess in der geforderten Weise beschreiben zu können (reproduzierbar, automatisierbar, verständlich, nachnutzbar und flexibel).

Grenzen Die Forderung nach einer vollständigen Beschreibung ist in der Praxis durch ein Computersystem allerdings kaum zu erfüllen. Die Automatisierbarkeit stößt beispielsweise bei Aktivitäten an ihre Grenzen, an denen menschliche Ressourcen beteiligt sind. Die Flexibilität einer Aktivität ist dadurch begrenzt, dass sie noch einen spezifischen Nutzen haben muss. So wäre es beispielsweise fraglich, eine Aktivität als Container für andere Aktivitäten zu definieren, die anhand eines Parameters zwischen einer sequentiellen oder parallelen Ausführung umschalten würde. Die Reproduzierbarkeit eines Workflows hängt von der Art und Weise ab, wie er Ergebnissdaten bei seiner Ausführung produziert (Provenance). Missier spezifiziert die Eingabe zur Ausführung eines Workflows als Tupel aus: Workflow(-Definition),

Eingabedaten des Workflows, externen Abhängigkeiten (*external dependencies*, ED) und dem Workflow-Management-System [MWHW12]. Unter ED werden hierbei jegliche Software-Artefakte zusammengefasst, die von Aktivitäten aufgerufen werden, aber extern, nicht unter der Kontrolle des WfMS sind (z. B. Programme, Bibliotheken und auch Datenbankzustände). Die Einbeziehung von Abhängigkeiten zu Software-Artefakten (WfMS und ED) ist bedeutsam, denn diese spielen eine entscheidende Rolle bei der Frage, wie welches Datum entstanden ist. Vergegenwärtigt man sich, dass nicht nur verschiedene Versionen dieser Software-Artefakte einzubeziehen sind, sondern auch, dass diese Artefakte wiederum Abhängigkeiten zu anderen Artefakten haben, so wird die hohe Komplexität der Aufgabe deutlich, solche Abhängigkeiten festzustellen und zu verwalten. Unter Einbeziehung solcher Daten kann man sich der Herausforderung stellen, zu entscheiden, ob ein Ablauf eines Workflows eine Reproduktion eines anderen Ablaufes ist. Doch dies ist ein eigenes, aktuelles Forschungsthema.

7.5.2. Nicht-funktionale Anforderungen

Im folgenden werden einige wichtige, nicht-funktionale und grundlegende Anforderungen an ein Experimentier-Computersystem formuliert, ohne einen Anspruch auf Vollständigkeit zu erheben. Die generelle Maxime lautet, sparsam mit Ressourcen umzugehen, wie beispielsweise der Zeit des Experimentators, CPU-Zeit, Speicherbedarf und Datenübertragung. Wissenschaft und insbesondere Experimentieren wird üblicherweise in Teamarbeit betrieben. Daher muss ein Experimentier-Computersystem grundsätzlich mehrbenutzerfähig sein. Neben den dafür notwendigen AKID-Eigenschaften und der formalen, vollständigen Beschreibung des Experimentier-Prozesses sind wichtige, nicht-funktionale Anforderungen eines Experimentier-Computersystems:

Blockierungen vermeiden Software-Komponenten wie Editoren, Repositorien und Experimentausführung dürfen sich nur für eine möglichst kurze Zeit gegenseitig blockieren. Hierfür sollen beispielsweise nur unbedingt notwendige Lese-/Schreibsperren für Teile des Experimentier-Workflows angelegt werden.

Laufzeit-Performanz Durch die parallele Ausführung von Experimentier-Workflow-Aktivitäten kann die Laufzeit des Workflows verkürzt werden. Im einfachsten Fall wird hierbei die Entscheidung, welche Aktivitäten parallelisierbar sind, dem Experimentator überlassen.

Speicherbedarf und Datenübertragung Ist das Experimentier-Computersystem als verteiltes System umgesetzt, bei dem zumindest das Experiment-Repositorium und die MUX-Programmausführung auf verschiedenen Computern betrieben werden, so spielt die zwischen diesen Komponenten benötigte Datenübertragungsmenge eine Rolle. Eine mehrfache Übertragung der Eingabedatenmenge für die MUX-Programmausführung sollte nach Möglichkeit vermieden werden (z. B. durch geeignetes Zwischenspeichern).

Zugriffsschutz und Datenintegrität In einem Mehrbenutzersystem wie dem Experimentier-Computersystem müssen den Benutzern (Experimentatoren) Rechte für das Lesen und Schreiben von Experimentier-Workflows erteilt werden können. Nach Instanziierung und Ausführung der Experimentier-Workflow-Definition muss diese vor Veränderung geschützt sein.

7.5.3. Vergleich existierender Systemklassen

In Tabelle 7.1 wird ein Vergleich von Scientific-Workflow-Systemen und Simulationssystemen mit Experimentierunterstützung vorgenommen, hinsichtlich der bisher aufgestellten Anforderungen an die Beschreibung von Experimentier-Prozessen und an ein Experimentier-Computersystem, das diese Anforderungen umsetzt.

Eigenschaft	**Scientific-Workflow-Systeme**	**Simulationssysteme**
Automatisierbarkeit	Ja.	Eingeschränkt.
Transparenz	Schwächen bei Verständlichkeit, da Ziel des Experiments versteckt.	Nachnutzbarkeit nur innerhalb des jeweiligen Simulationssystems.
AKID	I. A. nicht. Insbesondere keine Konsistenz der Artefakte.	Nein.
Domänenspezifische Konzepte	Nein.	Nein.
Formale Beschreibung	Ja.	Ja.

Tabelle 7.1.: Vergleich von Scientific-Workflow-Systemen und Simulationssystemen mit Experimentierunterstützung hinsichtlich der aufgestellten Anforderungen an ein Experimentier-Computersystem

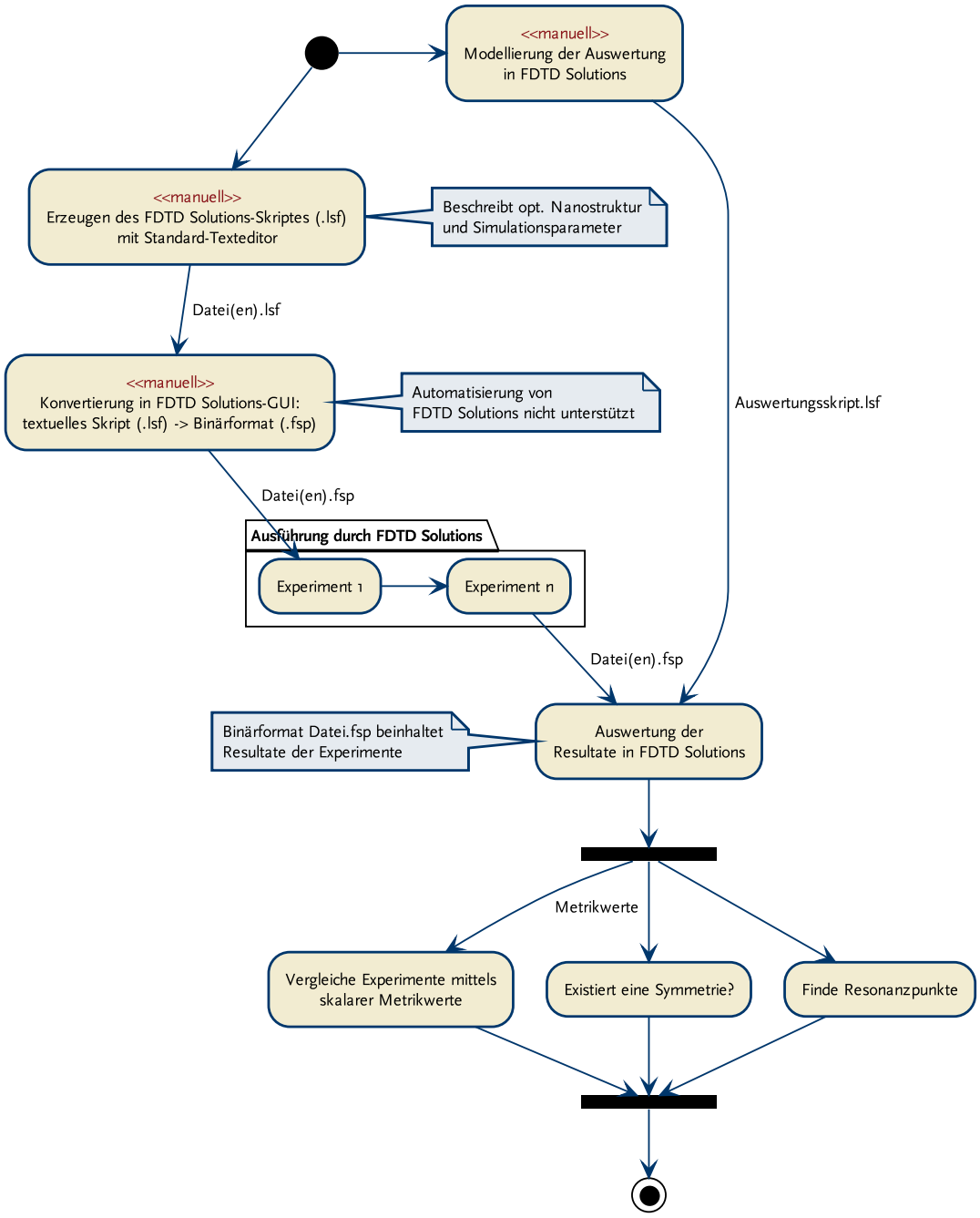


Abbildung 7.4.: Traditioneller Experimentier-Prozess mit optischen Nanostrukturmodellen mit Sicht der verwendeten Artefakte (als UML-Aktivitätsdiagramm)

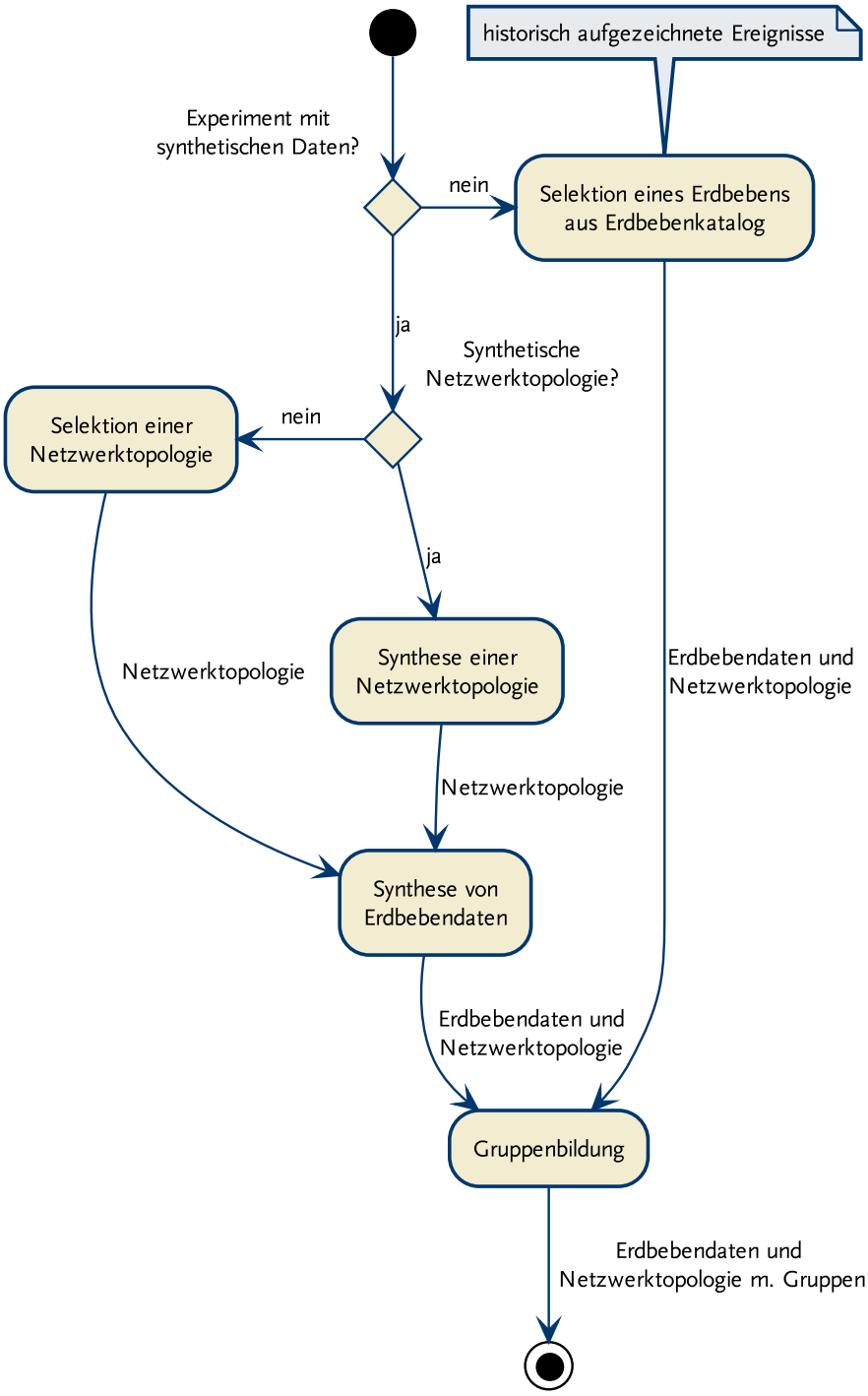


Abbildung 7.6.: Teilprozess Datenaquise des Experimentier-Prozesses in SOSEWIN zur Bereitstellung von Erdbebendaten und Netzwerktopologien (als UML-Aktivitätsdiagramm)

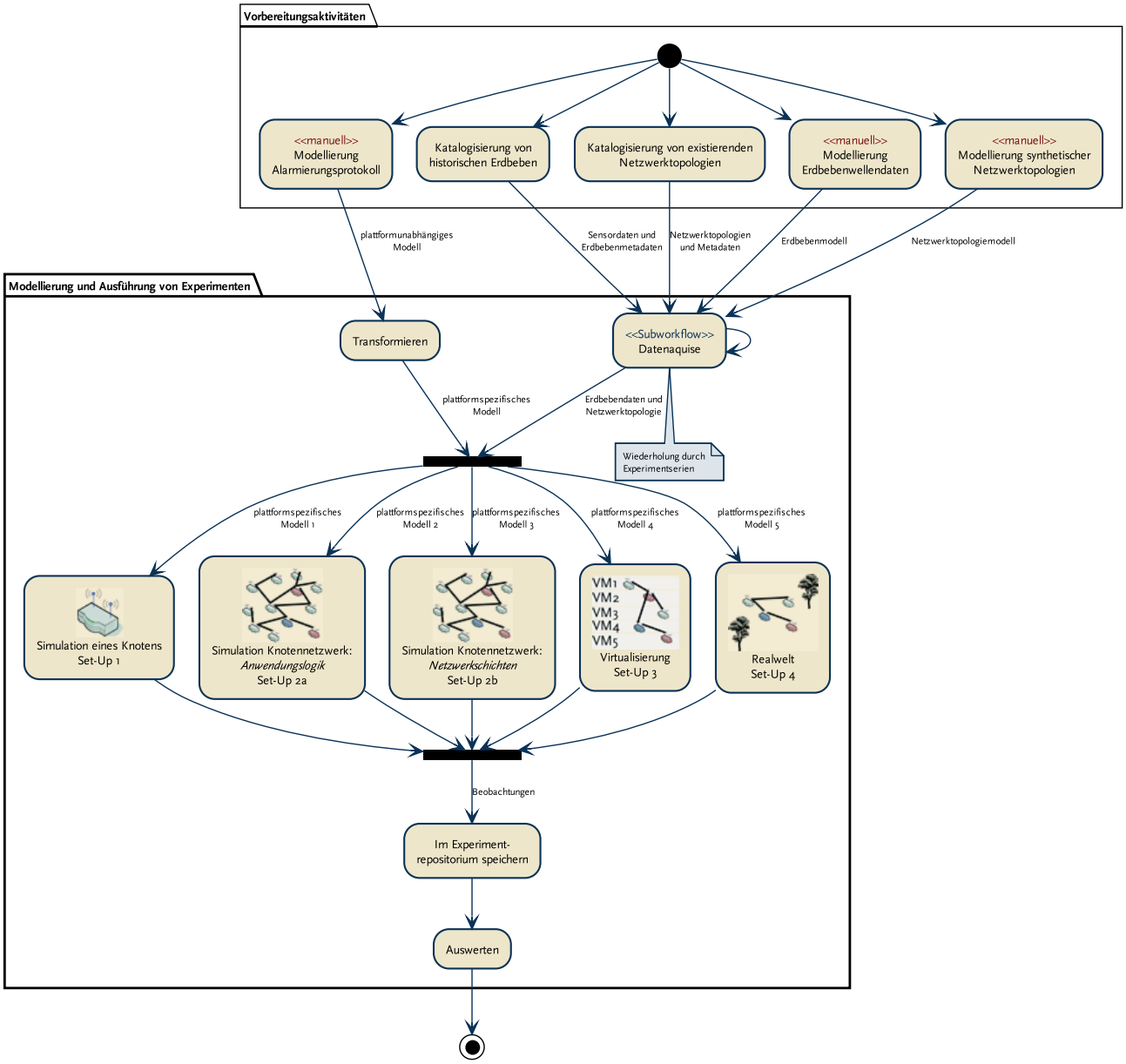


Abbildung 7.7.: Experimentier-Prozess in SOSEWIN (inklusive vorbereitenden Aktivitäten und des Teilprozesses Datenaquise aus Abbildung 7.6) zur Umsetzung des modellbasierten Ansatzes, dargestellt in Abbildung 7.5 (als UML-Aktivitätsdiagramm)

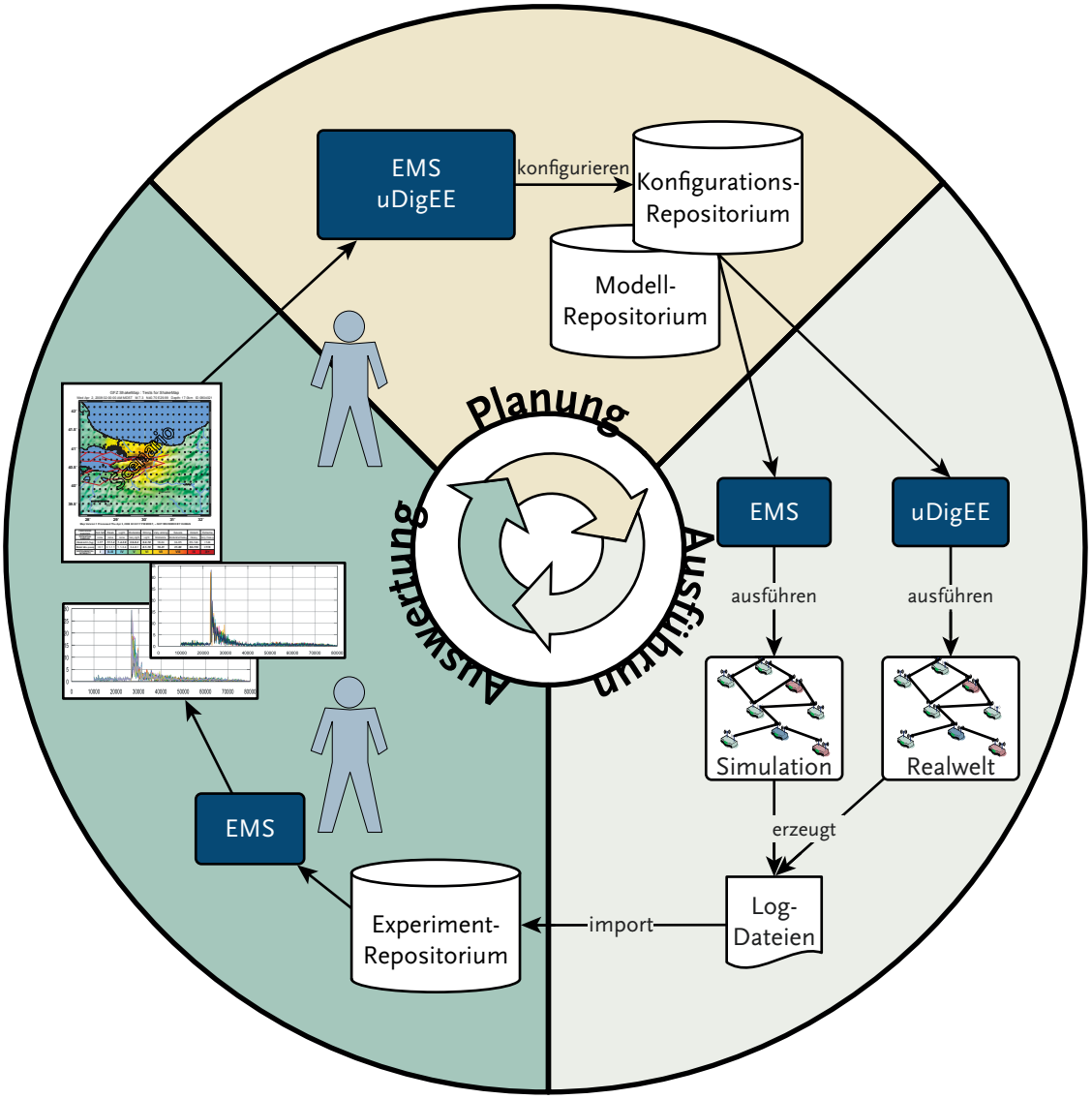


Abbildung 7.8.: Werkzeuge und Artefakte im Experimentier-Prozess in SOSEWIN

8. Konzeption der Experimentier-Workflow-Beschreibungssprache ExpL

In diesem Kapitel 8 werden Konzepte und deren Umsetzung durch Sprachelemente der in dieser Arbeit entwickelten Experimentier-Workflow-Beschreibungssprache *Experimentation-Language (*ExpL*)* dargestellt. Sie ist ein Bestandteil des ExpL-Workflow-Systems. Das ExpL-Workflow-System ist ein Softwaresystem, das aus dem ExpL-*Workflow-Management-System (*ExpL-*WfMS)*, ExpL (in Form dessen Sprachmetamodells), der Notationssprache von ExpL und dazu konformen ExpL-Wf-Definitionen besteht. Es ist eine konkrete Ausprägung des in Abschnitt 7.5.1 dargestellten Experimentier-Computersystems und erfüllt die an ein solches System gestellten Anforderungen (siehe Abschnitt 7.5). Abbildung 8.1 auf der folgenden Seite visualisiert hierbei die Zusammenhänge zwischen einem Experimentier-Computersystem und dem ExpL-Workflow-System. Dabei werden Experimentier-Prozesse zunächst als Experimentier-Workflows unabhängig von einer konkreten Notationsform gefasst. Ein ExpL-*Workflow (*ExpL-*Wf)* ist ein Experimentier-Workflow, der konkrete, formale Beschreibungsmittel in Form des ExpL-Sprachmetamodells verwendet. Die Menge der durch das ExpL-Wf-System unterstützten ExpL-Wf-Definitionen wird durch die formale Workflow-Beschreibung in Form des ExpL-Sprachmetamodells gebildet.

Im Folgenden wird auf die domänenspezifischen Konzepte und Bestandteile des ExpL-Sprachmetamodells eingegangen, welche an exemplarischen ExpL-Wf-Definitionen gezeigt werden. Kapitel 9 zeigt die technische Architektur des ExpL-Wf-Systems.

8.1. Ein sprachzentrierter Ansatz

Jede MUX-Domäne hat bestimmte, spezifische Anforderungen bezüglich Konfiguration, Ausführung und Auswertung von Experimenten. Gleichwohl bleibt die Domäne Experimentieren mit computerbasierten Modellen als Gemeinsamkeit. Deshalb sieht der ExpL-Ansatz vor, das Prinzip von *separation of concerns* anzuwenden. Die über MUX-Domänen gleichbleibenden Experimentier-Basiskonzepte werden durch die Sprache ExpL-*Kern* beschreibbar und für die Konzepte aus den Aspekten Konfiguration, Ausführung und Auswertung stehen separate, gekoppelte DSLs zur Verfügung (Sprachkopplung).

Theisselmann führte hierfür den Begriff *sprachzentrierter Ansatz* ein – *Language-Centered-Approach (LCA)* – für die Kombination verschiedener, ergänzender DSLs auf der Basis eines

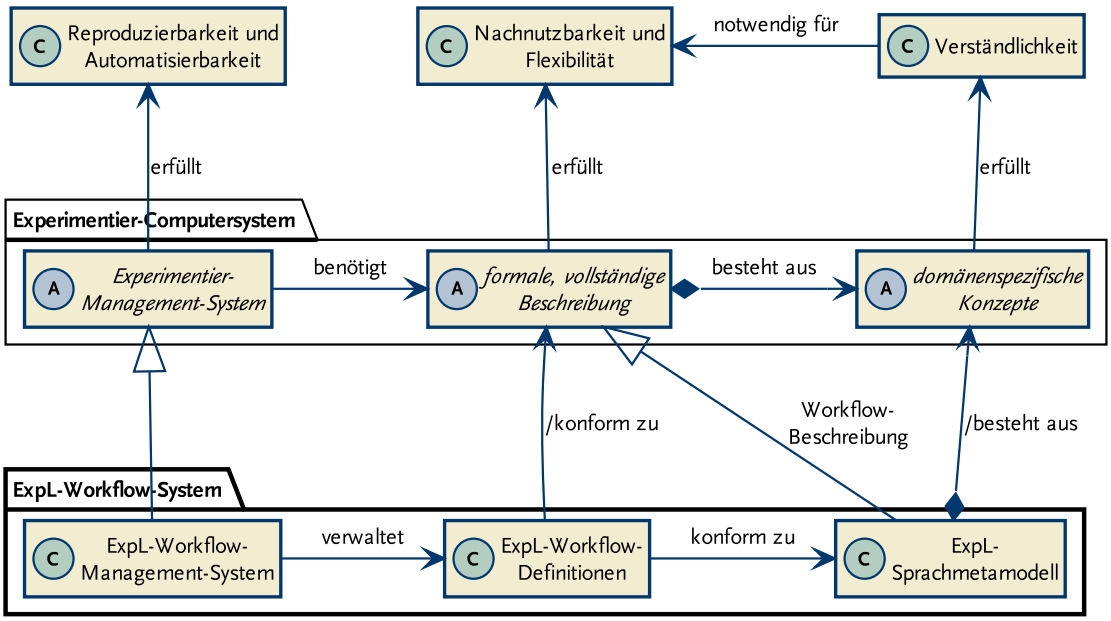


Abbildung 8.1.: Relationen zwischen Anforderungen an ein Experimentier-Computersystem und deren Umsetzung durch das ExpL-Workflow-System (als UML-Klassendiagramm; erweitert Abbildung 7.10 auf Seite 93)

gemeinsamen Sprachmetamodells [TKK+10]. LCA ist ein Grundprinzip des ExpL-Ansatzes und wird durch Abbildung 8.2 illustriert.

ExpL-Kern stellt Experimentier-Basiskonzepte in Form von Workflow-Konzepten (*Aktivität, Ressource, Artefakt* und *Kontrollfluss*) und *deklarativen Sprachelementen* bereit (darauf wird in Abschnitt 8.3 eingegangen). ExpL-Kern stellt außerdem die Verbindung zu den ergänzenden drei Sprachen *Konfigurations-DSL, Ausführungs-DSL* und *Auswertungs-DSL* her. Dabei bestehen funktionale und strukturelle Abhängigkeiten zwischen diesen Sprachen, so dass keine dieser Sprachen sinnvoll separat verwendet werden kann (und soll). Das Konzept, diese vier Sprachen zusammen zu verwenden, wird fortan als ExpL-*Sprachfamilie* bezeichnet werden. Eine ExpL-Sprachfamilie für eine konkrete MUX-Domäne wird verkürzt als ExpL benannt, so dass dieser Begriff weiterhin für die gesamte, konkrete Sprachbeschreibung (der gekoppelten DSLs) verwendet wird. Die ExpL-Sprachfamilie besteht dabei aus externen DSLs.

Durch die Aufteilung in sich ergänzende DSLs wird deren Austauschbarkeit ermöglicht (austauschbar mit Ausnahme von ExpL-Kern) und dadurch sowohl die Anpassbarkeit an eine bestimmte MUX-Domäne als auch ihre Wiederverwendung in einer anderen MUX-Domäne gefördert. Austauschbarkeit und Wiederverwendung sind hierbei wichtige Argumente für den sprachzentrierten Ansatz, die durch Konzepte wie Vererbung und Relationen unterstützt werden (siehe Sprachbeschreibung durch OOMM auf Seite 66). Scientific-Workflow-Management-Systeme bieten demgegenüber für eine Anpassung an eine MUX-Domäne lediglich die Möglichkeit, spezifische Aktivitäten zu implementieren; typischerweise ohne die Möglichkeit strukturierter Datentypen. Die Funktionalitäten der Sprachen in der ExpL-Sprachfamilie sind typischerweise verschieden stark ausgearbeitet, um den Erfordernissen in der entsprechenden

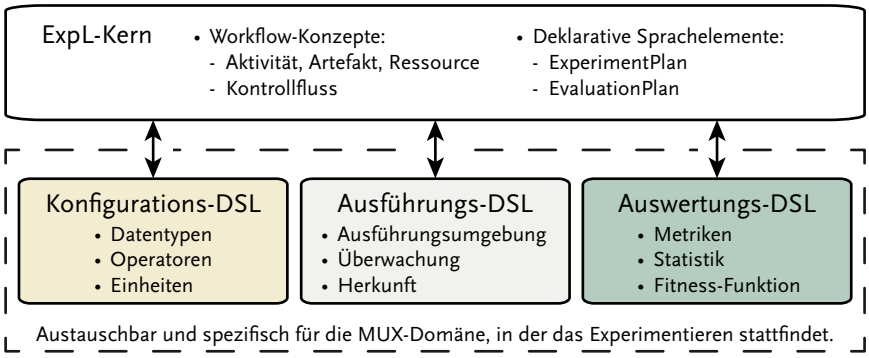


Abbildung 8.2.: ExpL-Sprachfamilie unter Anwendung eines sprachzentrierten Ansatzes (LCA)

MUX-Domäne zu genügen.

Konfigurations-DSL Eine Konfigurations-DSL stellt Datenstrukturen und -typen, Einheiten und Operatoren bereit, die für die jeweilige MUX-Domäne innerhalb der Planungsphase benötigt werden. Dies sind in der SOSEWIN-Fallstudie beispielsweise spezielle, komplexe Datentypen (z. B. für die Netzwerktopologie), georäumliche Datentypen (z. B. für die Position von Knoten) und Operatoren, die eine georäumliche Variation beschreiben können (z. B. zur Variation der Distanz zwischen Hypozentrum und SOSEWIN). In der Nano-Fallstudie können Monitore als Bestandteil einer Konfigurations-DSL betrachtet werden. Auch die in ExpL integrierten Beschreibungsmöglichkeiten zur Formulierung von Beschränkungen (*constraints*) an Modelleingabeparameterbelegungen zur Einschränkung des Parameterraumes gehören zur Konfigurations-DSL.

Ausführungs-DSL Eine Ausführungs-DSL dient hauptsächlich der Beschreibung der möglichen Ausführungsumgebungen des MUX innerhalb der Ausführungsphase. Ausführungsumgebungen liegen beispielsweise im Simulationskontext und in der Realwelt (beide Arten sind in der SOSEWIN-Fallstudie anzutreffen; in den Fallstudien SLEUTH und Nano wird der Simulationskontext genutzt). Eine Ausführungs-DSL beschreibt:

- wie die Modelleingabeparameterwerte für das MUX und die Laufzeitparameterwerte an die Ausführungsumgebung übermittelt werden,
- wie das MUX-Programm ausgeführt wird,
- welche (temporären Zwischen-)Artefakte evtl. erzeugt werden müssen und
- wie die Ausführung zu überwachen ist.

Auswertungs-DSL Eine Auswertungs-DSL umfasst Ausdrucksmittel zur Formulierung von Metrik- und Statistikfunktionen über Werte, die durch Experimentläufe gewonnen werden (Modellausgabewerte und Monitoring-Informationen). Sie umfasst auch Ausdrucksmittel für

die Anwendung von Optimierungsverfahren und beschreibt die semi- und vollautomatische Auswertungsphase. Ein Beispiel für eine Auswertungs-DSL ist die GIS-DSL in der SLEUTH-Fallstudie.

8.2. Grundprinzipien

Das Prinzip der *separation of concerns* wird in der Informatik häufig angewendet und bedeutet die Trennung von Konzepten nach ihren Aufgaben.[1] In der Modellierung wird das Prinzip beispielsweise angewendet, um Simulationsmodell und Simulationskern voneinander zu trennen (siehe Abschnitt 2.2.1). In der ExpL-Sprachfamilie wird *separation of concerns* außer beim LCA fortgesetzt angewendet, um Nachnutzbarkeit, Flexibilität und Verständlichkeit zu erhöhen. Im folgenden werden die Grundprinzipien (neben des LCA) für das Design von ExpL erläutert.

Trennung Exp-Wf-Phasen Die Experimentier-Workflow-Phasen *Planung*, *Ausführung* und *Auswertung* werden in ExpL durch separate Sprachelemente repräsentiert. Für die Planung ist dies hauptsächlich das Sprachelement `ExperimentPlan` für die Ausführung `RunSpecification` und für die Auswertung `EvaluationPlan`. Durch diese geradlinige Übertragung der Konzepte, die der Experimentator bereits kennt und verwendet, in die Sprache ExpL wird deren Verständlichkeit erhöht.

Trennung Deklaration Definition Alle Ressourcen und Artefakte (z. B. das MUX) werden deklariert (Wert nicht zugewiesen), bevor sie in Aktivitäten definiert (Wert zugewiesen) und verwendet werden können. Dies geschieht explizit und separat von den Aktivitätsdefinitionen. Dadurch wird auch eine gewisse Trennung zwischen Struktur und Verhalten (auf Basis der definierten Strukturen) erreicht.

Artefakttrennung Die Beschreibungen der Artefakte MUX, MUX-Programm, Ausführungsumgebung und Experiment erfolgt mit Hilfe separater ExpL-Sprachelemente, die nur durch Referenzen zueinander in Beziehung stehen. Dadurch lässt sich beispielsweise die Beschreibung der Ausführungsumgebung austauschen, ohne die Beschreibung des MUX verändern zu müssen. Das bedeutet Flexibilität und fördert die Nachnutzbarkeit.

Konsistenz und Persistenz Als Anforderungen an ein Computersystem für das Experimentier-Management wurden die AKID-Eigenschaften genannt (Abschnitt 7.5), die den Begriff der *Konsistenz* beinhalten. An dieser Stelle soll der Begriff noch auf einer semantischen Ebene gefasst werden, indem gefordert wird, dass für Workflow-relevante Daten (z. B. Werte von

[1] Weitere Anwendungen von *separation of concerns* sind beispielsweise alle Arten von Schichtenmodellen und Programmierparadigmen, wie z. B. das Muster *Model-View-Controller (MVC)*.

Modelleingabeparametern) stets der Kontext ihrer Herkunft nachvollzogen werden kann. Zusammen mit dem Grundprinzip „Trennung von Deklaration und Definition“ bedeutet dies, Workflow-relevante Daten müssen auf eine Deklaration in der Workflow-Definition zurückzuführen sein. Verweise auf externe, nicht durch das ExpL-WfMS gespeicherte Ressourcen und Artefakte sollen möglichst durch *Persistente-Identifier (PI)* erfolgen (aus Gründen der Migration werden auch Methoden ohne PI angeboten, siehe Abschnitt 8.3.1). Durch PI ist sichergestellt, dass das Verweisziel unverändert dem entspricht, wie es zum Zeitpunkt des Erstellens des Verweises gewesen ist.

8.3. Sprachelemente von ExpL

Die Sprachbeschreibung von ExpL liegt als Metamodell vor und definiert dessen Syntax, die aus Sprachkonstruktdefinitionen besteht (siehe Abschnitt 6.2.1 auf Seite 61). Diese konkrete Sprachbeschreibung wird als ExpL-*Sprachmetamodell* bezeichnet. Eine ExpL-Wf-Definition ist somit eine Sprachinstanz unter Verwendung der konkreten Sprachkonstruktdefinitionen aus dem ExpL-Sprachmetamodell.

Zudem wurde eine Notationssprache für ExpL erstellt, mit dessen Hilfe textuelle Repräsentationen von ExpL-Sprachinstanzen notiert werden. Diese Notationssprache wird durch eine kontextfreie Grammatik (siehe Abbildung B.1 im Anhang) beschrieben und ist benannt als *Notationssprache von ExpL*. Dadurch lassen sich Sprachinstanzen der Notationssprache formulieren (Zeichenketten, die konform hinsichtlich der zugehörigen Grammatik sind), welche die gleiche Semantik haben, wie die entsprechende Sprachinstanz in ExpL (in Form eines Modells, das konform zum ExpL-Sprachmetamodell ist).

Sowohl die metamodellbasierte Sprachbeschreibung von ExpL als auch die der Notationssprache von ExpL bestehen aus Sprachkonstruktdefinitionen, welche durch eine definierte Abbildung in Relation stehen (siehe Listing B.1 auf Seite 176 im Anhang). Struktur, Anzahl und Benennung der Sprachkonstruktdefinitionen beider Sprachen können sich unterscheiden. Dies ist beispielsweise sinnvoll, um die textuelle Repräsentation an einigen Stellen lesbarer zu gestalten. Zentrale Sprachkonstruktdefinitionen (wie beispielsweise Arten von Ressourcen oder Artefakten) sind in beiden Sprachen gleich benannt und sollen im Folgenden als *Sprachelemente* (von ExpL) zusammengefasst werden. Die Namen von Sprachelementen sind im Text dieses Kapitels in runden Klammern angegeben. Soll explizit auf eine Sprachkonstruktdefinition der Notationssprache von ExpL verwiesen werden, so wird dies im Text durch Voranstellen des Bezeichners „Schlüsselwort“ erreicht.

Alle Sprachelemente von ExpL sind unterhalb eines Wurzelsprachelementes organisiert, dem `ExperimentationWorkflow`. Dieses Element erlaubt es, Eigenschaften zu formulieren, die für den gesamten ExpL-Wf bedeutsam sind. Hierzu zählt insbesondere eine Beschreibung des Untersuchungsziels (im Attribut `description`). Sie erfolgt als Freitext. Der Grundgedanke dabei ist es, für einen Experimentier-Workflow ähnliche Angaben zu dessen Zweck und Grenzen zu machen, wie es mit dem *Wortmodell* von Bossel für Simulationsmodelle gedacht ist [Bos94].

Deklarative Sprachelemente Deklarative Computersprachen[2] definieren die Logik zum Erreichen eines gewünschten Zieles oder Resultates auf der Basis von Funktionen und Operatoren (*funktionale Programmiersprachen*), logischer Kalküle (z. B. Lambda-Kalkül, Prädikatenlogik) oder Datenflussgraphen. Im Gegensatz zu imperativen Programmiersprachen definieren sie somit den Kontrollfluss nicht in Form von Sequenzen einzelner Anweisungen, die in ihrer Abfolge zu dem gewünschten Resultat führen (sollen). ExpL ist keine deklarative Sprache, denn in ihr wird Kontrollfluss durch die Komposition von Aktivitäten definiert, die eine Abfolge von Anweisungen darstellt. ExpL besitzt aber deklarative Sprachelemente, mit denen der Experimentator auf einer Menge von MUX sein Experimentierziel formulieren kann. Die beiden deklarativen Sprachelemente von ExpL sind `ExperimentPlan` und `EvaluationPlan`.

8.3.1. Ressourcen und Artefakte

Eine Ressource in ExpL ist konform zum Ressourcen-Begriff aus dem Workflow-Kontext, mit der Einschränkung, dass menschliche Ressourcen nicht berücksichtigt sind, da sie für die Automatisierung des Experimentier-Workflows durch das ExpL-Workflow-System keine Rolle spielen. In ExpL sind Ressourcen somit ausschließlich maschineller Art in Form von Computersystemen (auf dem Aktivitäten ausgeführt werden können) und Repositorien (zur Datenspeicherung). Abbildung 8.3 zeigt Ressourcen und Artefakte als Klassen im ExpL-Sprachmetamodell.

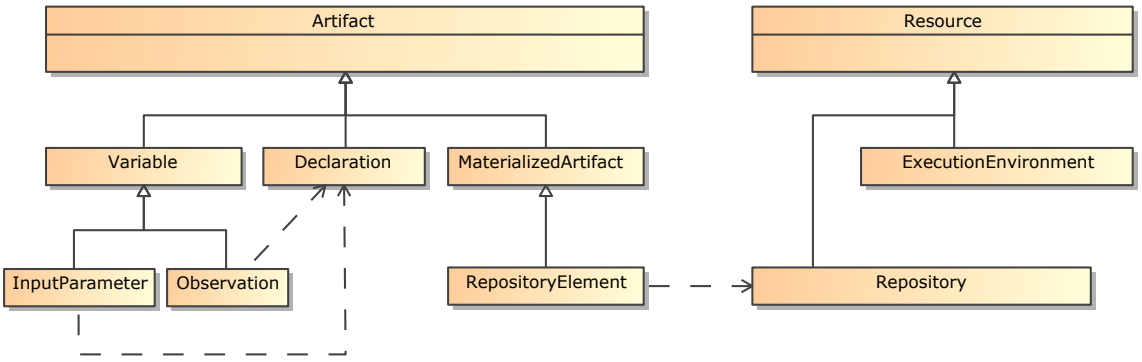


Abbildung 8.3.: Ausgewählte Ressourcen und Artefakte im ExpL-Sprachmetamodell (Auszug als UML-Klassendiagramm)

Repositorien in ExpL speichern *Artefakte*, die entweder für eine Aktivitätsausführung notwendig sind oder die durch eine Aktivität erzeugt wurden. Repositorien sind danach geordnet, ob sie Artefakte in mehreren Versionen verwalten können (versioniert) oder nicht (nicht-versioniert). Ein Beispiel für ein nicht-versioniertes Repositorium[3] ist das klassische Dateisystem (`FSRepository`), und für ein versioniertes Repositorium sind im ExpL-Ansatz Quell-

[2] Bekannte deklarative Sprachen sind beispielsweise Haskell, LISP und Prolog.

[3] Ein nicht-versioniertes Repositorium widerspricht dem Grundprinzip der Persistenz, weshalb dessen Verwendung nicht empfohlen wird. Zur leichteren, schrittweisen Migration von einem Experimentieren ohne Werkzeugunterstützung kann es aber hilfreich sein.

textverwaltungssysteme wie *Apache Subversion* [sub11, PCSF08] und *git* [TH05] vorgesehen. Das ExpL-WfMS beinhaltet ein (spezielles) *Konfigurations-Repositorium* zur Speicherung von ExpL-Wf-Definitionen und deren Instanzen zur Laufzeit der Workflows. Innerhalb einer ExpL-Wf-Definition werden typischerweise Referenzen auf mindestens ein externes *Modell-Repositorium* und ein externes *Ergebnis-Repositorium* gespeichert (zu Repositorien siehe Abschnitt 9.3.6).

Ein Artefakt kann in Form einer Datei[4] `(MaterializedArtifact)` vorliegen oder in feingranularerer Form als bedeutungstragende Struktur[5], wie beispielsweise einer Variablendeklaration `(Declaration)` oder einer Variablenbelegung `(Variable)`.

MUX-Deklaration

Ein zentrales Artefakt ist die MUX-Deklaration zur Beschreibung von Modelleingabeparametern `(InputParameterDeclaration)` und Modellausgabeports `(ObservableDeclaration)` des MUX. Eine solche Beschreibung enthält für jedes dieser Ein- und Ausgabedeklarationen eine Datentypdefinition (mit optionaler Einschränkung auf einen Wertebereich) und optional eine Standardwertebelegung (Schlüsselwort `default`). Modellausgaben können auch als Artefakten `(ObservableArtifactDeclaration)` deklariert werden, um beispielsweise die Erzeugung einer Datei durch das MUX zu beschreiben. Ein- und Ausgabedeklarationen lassen sich zu benannten Gruppen zusammenfassen `(DeclarationGroup)`. Zudem kann in einer MUX-Deklaration die verwendete Modellbeschreibungssprache, der Modellname und optional eine Beschreibung (in Freitext) zur Dokumentation angegeben werden.

Beispiel Listing 8.1 auf der folgenden Seite zeigt die verkürzte, textuell notierte MUX-Deklaration im Bossel-Fischfang-Beispiel (siehe Abschnitt 2.3.2; ein weiteres Beispiel aus der SLEUTH-Fallstudie ist im Anhang in Listing C.2 auf Seite 184). Dabei werden Standard-Initialwerte für die Modelleingabeparameter (des MUX) angegeben, die aber pro Experiment geändert werden können (z. B. für Boote, Zeile 6). Zudem werden potentielle Modellausgaben deklariert, wobei Zeitreihen möglich sind: Pro Wert (des angegebenen Typs) kann eine Modellzeit gespeichert werden.[6] Somit kann die Entwicklung der Anzahl der Boote über die Zeit gespeichert werden (Zeile 24).

---

[4] In Form einer Datei ist das Artefakt in ExpL analog dem Modellelement *Artifact* aus dem UML-Verteilungsdiagramm.

[5] Die Bereitstellung von bedeutungstragenden Strukturen muss vom Typ des Repositoriums unterstützt werden. ExpL bietet zwei solcher Repositorientypen: Eines für metamodellbasierte Strukturen `(CDORepository)` und eines für relationale Datenbanken `(RDBMSRepository)`.

[6] Dadurch ergibt sich eine Diskretisierung der Modellausgabewerte, die in der Darstellung im Buch aufgrund der Verwendung einer interaktiven GUI nicht vorkommt. An dieser Stelle wird das Beispiel entsprechend angepasst, was in Abschnitt 8.3.5 auf Seite 114 weiter beschrieben wird.

```
GitRepository modelRepository {
  ModelRepository URI = /* ... */ workingDirectory = ModelRepositoryLocalCache
  elements {
    GitModelRepositoryElement "FISHFANG.MOD" {
      modelName "FISCHFANG-DYNAMIK" inputParameters {
        InputParameterDeclaration Boote as ExpInteger { default "100" }, /* ... */
      }
      groups {
        DeclarationGroup Festparameter {
          inputParameters {
            InputParameterDeclaration Fanggebiet as ExpReal
              { default "100.0", unit "km2" }, /* ... */
          }
        },
        DeclarationGroup Systemparameter { /* ... */ },
        DeclarationGroup Szenarioparameter {
          inputParameters {
            InputParameterDeclaration maxSpezFangmenge as ExpReal
              { default "100.0", unit "t Fisch/(Boot.Jahr)" }, /* ... */
          }
        }
      }
      observables {
        ObservableDeclaration Boote as ExpReal, /* Zeitreihe möglich */
        ObservableDeclaration Fische as ExpReal, /* Zeitreihe möglich */
        ObservableDeclaration Fangmenge as ExpReal /* Zeitreihe möglich */
      }
    }
  }
}

Legende: Schlüsselwort, Identifier, Referenz, "String", /* Kommentar */
```

Listing 8.1: Verkürzte, textuell notierte MUX-Deklaration am Beispiel Bossel-Fischfang [Bos94, S. 169 ff.] (vollständige Deklaration in Listing B.2 auf Seite 177)

Typsystem

ExpL besitzt ein explizites Typsystem (siehe Auszug des ExpL-Metamodells im Anhang in Abbildung A.5 auf Seite 171), mit welchem die Datentypen von Modelleingabeparameter und Modellausgabeports zur Entwurfszeit in der MUX-Deklaration beschrieben werden. Es umfasst einfache Basisdatentypen (z. B. Real, String, Integer, Boolean), Wertemengen und kann um komplexe Datentypen erweitert werden. Exemplarisch für komplexe Datentypen beinhaltet ExpL `Point2d` und `Point3d`, mit dem Punkte in einem georeferenzierten Koordinatensystem angegeben werden können. Die Typisierung in ExpL ist statisch, so dass Typüberprüfungen vor Ausführung der Workflow-Instanz vorgenommen werden (z. B. ob ein MUX bzgl. seiner Eingabeparemeterdatentypen in einer bestimmten Ausführungsumgebung

ausgeführt werden kann).[7] Zudem kann zu jeder Variablendeklaration (`Declaration`) eine SI-Einheit als Beschreibung hinzugefügt werden (Schlüsselwort `unit`).[8]

8.3.2. Aktivitäten

Eine Aktivität in ExpL ist konform zum Aktivitätsbegriff aus dem Workflow-Kontext. Sie wird im ExpL-Sprachmetamodell als *Task* bezeichnet. Abbildung 8.4 zeigt die verschiedenen Aktivitäten und ihre Hierarchie in Form von Klassen des ExpL-Sprachmetamodells.

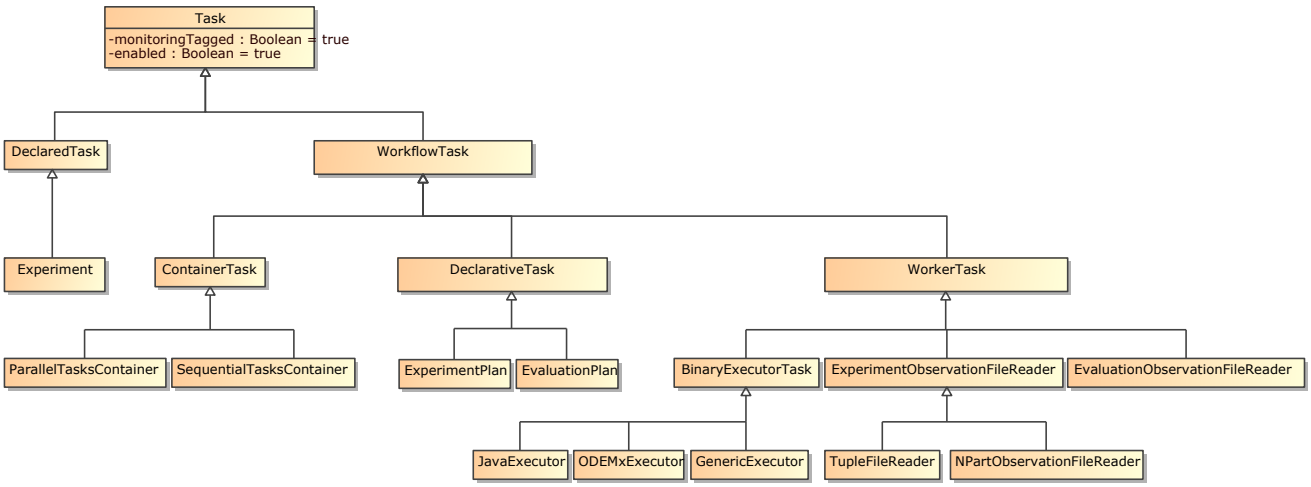


Abbildung 8.4.: Hierarchie der Aktivitäten (*tasks*) im ExpL-Sprachmetamodell (Auszug als UML-Klassendiagramm)

In ExpL werden ausschließlich Kontrollflüsse zwischen Aktivitäten definiert. Die Spezifikation von Datenflüssen ist nicht notwendig, da alle Aktivitäten auf einen gemeinsam genutzten Speicher zugreifen, welcher die Daten sowohl der ExpL-Wf-Definition als auch der -Instanz enthält (siehe Abschnitt 9.2). Die Interaktion mit Workflow-Applikationen, die per se keinen Zugriff auf den gemeinsamen Speicher haben, ist über spezielle Aktivitäten geregelt.

Aktivitäten in ExpL werden danach gruppiert, in welcher Phase sie definiert werden: Zur Entwurfszeit oder zur Laufzeit. So werden Ausprägungen der Aktivität `WorkflowTask` vom Workflow-Designer in der ExpL-Wf-Definition zur Entwurfszeit spezifiziert. Demgegenüber steht ein `DeclaredTask` als einzige Aktivitätsklasse, die üblicherweise durch das ExpL-WfMS automatisch zur Laufzeit abgeleitet wird (indem es den zur Entwurfszeit spezifizierten `Decla-`

[7] Die statische Typüberprüfung ist exemplarisch auf der Basis von Modell-Checking implementiert (siehe Seite 123).

[8] Eine statische Überprüfung der Einheiten des *Système international d'unités (SI)* wäre wünschenswert, ist aber gegenwärtig nicht implementiert. Zudem könnten SI-Einheiten automatisch ineinander umgerechnet werden. Dadurch ließen sich Fehlerquellen durch inkompatible SI-Einheiten oder falsche Größenordnungen reduzieren.

rativeTask verwendet).[9] Die Ausprägungen der Aktivität DeclarativeTask stellen *deklarative Sprachelemente* dar (Experimentplan und Auswertungsplan).

Ein ContainerTask beinhaltet andere Aktivitäten, wobei durch seine zwei Ausprägungen festgelegt ist, ob diese Aktivitäten parallel oder sequentiell ausgeführt werden sollen. Durch Austausch einer der beiden konkreten Ausprägung gegen die andere in der Workflow-Definition ist ein leichter Wechsel des Workflow-Kontrollflussmusters vom *sequentiellen* zum *parallelen Ablauf* (und umgekehrt) möglich, ohne die beinhalteten Aktivitäten verändern zu müssen. Verschachtelungen von ContainerTask sind möglich. Ein paralleler Ablauf kann beispielsweise wiederum mehrere sequentielle Abläufe beinhalten (und somit parallelisieren). Alle Aktivitäten innerhalb eines parallelen Ablaufs müssen beendet sein, bevor im Workflow nachfolgende Aktivitäten ausgeführt werden.

Alle Ausprägungen von WorkerTask klassifizieren Aktivitäten, die weder zu ContainerTask noch zu DeclarativeTask zugeordnet werden können. Dies sind insbesondere kleinere Aufgaben, wie beispielsweise das Lesen und Schreiben von Ein- und Ausgabedateien.

Aktivitätsüberwachung

Für jede Aktivität kann angegeben werden, ob ihre Ausführung durch das ExpL-WfMS überwacht werden soll. Dabei können pro Aktivität folgende *Monitoring-Informationen* aufgezeichnet werden (ExpL-Sprachmetamodellelement RunInformation, siehe Abbildung A.3 auf Seite 170):

- Start- und Stopzeit (Realzeit),
- Änderungen der eigenen ExpL-Wf-Instanz (siehe Provenance in Abschnitt 9.2),
- CPU-Nutzung und Speicherbedarf des MUX-Programmes während seiner Ausführung.

8.3.3. Experimentplan

Das deklarative Sprachelement ExperimentPlan (siehe Abbildung 8.5) ist eine Beschreibung von Experimenten und deren Ausführung. Ein ExperimentPlan basiert auf einer Menge von zu untersuchenden Modellen (MUX), die eindeutig referenziert werden in Form von Artefakten in versionierten Repositorien (VersionizedRepositoryElement). Ein MUX-Artefakt kann hierbei gegen eine andere Version ausgetauscht werden, sofern die MUX-Deklaration bei beiden Versionen identisch ist, ohne weitere Änderungen am Experimentier-Workflow vornehmen zu müssen. Dadurch kann über einer Menge von MUX-Artefakten, die konform zur selben MUX-Deklaration sind, iteriert werden (z. B. anhand der Versionsnummer als PI

[9] Von DeclaredTask gibt es nur eine Ausprägung: Die Aktivität Experiment, welche aber auch zur Entwurfszeit spezifiziert werden kann. Dies ist beispielsweise sinnvoll, um eine einzelne Parameterwertebelegung außerhalb einer Experimentserie zu evaluieren. Es existieren jedoch weitere Bestandteile im ExpL-Sprachmetamodell, die nur durch das ExpL-WfMS erzeugt werden (z. B. Ausführungsüberwachung).

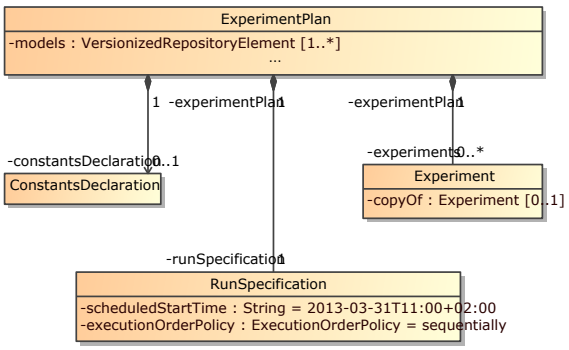


Abbildung 8.5.: Auszug des Klassendiagramms mit Sprachelement `ExperimentPlan` des ExpL-Sprachmetamodells (als UML-Klassendiagramm)

im entsprechenden Repositorium). Dadurch lassen sich Verhaltensänderungen in der Ausführung des MUX über dessen verschiedene Versionen einfach beobachten und Vergleiche werden ermöglicht.

Für jedes MUX ist in der MUX-Deklaration definiert, welche Modelleingabeparameter existieren, die für eine Ausführung initial mit Werten belegt werden müssen. Der `ExperimentPlan` gibt an, wie diese Werte berechnet werden: Entweder durch die Definition einer Konstante (`ConstantsDeclaration`) oder durch eine Variable (`ParameterVariationDeclaration`), die innerhalb einer Ausführungsspezifikation (`RunSpecification`) definiert wird.

Ausführungsspezifikation

Die Ausführungsspezifikation (`RunSpecification`) ermöglicht die Angabe, ob und wie Experimente wiederholt ablaufen sollen. Man kann sie als spezielle Form des Workflow-Kontrollflussmusters *Iteration* sehen, das in ExpL nur für Experimente definiert ist. Die Ausführungsspezifikation (`RunSpecification`) hat folgende konkrete Ausprägungen (siehe Klassendiagramm im Anhang in Abbildung A.4 auf Seite 171):

`SingleRunSpecification` Spezifiziert ein einzelnes Experiment (mit konstanten Parameterwerten) ohne Wiederholung.

`RepeatedMultiRunSpecification` Spezifiziert ein einzelnes Experiment (mit konstanten Parameterwerten) und `n` Wiederholungen (die zu `n` Experimentläufen führen).

`RangeMultiRunSpecification` Spezifiziert multiple Experimente durch Angabe von: Variationen für die Belegung der Modelleingabeparameter, Iteration über einer Menge von MUX-Versionen (mit identischer MUX-Deklaration) und Beschränkungen für den Parameterraum (siehe folgendem Abschnitt 8.3.6).

`RepeatedRangeMultiRunSpecification` Ist eine Spezialisierung von `RangeMultiRunSpecification` mit `n` Wiederholungen.

Parameterbelegungsvariation

Variationen für die Belegung der Modelleingabeparameter des MUX werden durch Spezialisierungen des Sprachelementes `ParameterVariationDeclaration` beschreibbar. Jede dieser Deklarationen definiert eine Parameterwertemenge PW_i, wobei i der Index des zugehörigen Modelleingabeparameters in der MUX-Deklaration ist. Es stehen hierbei folgende Sprachelemente zur Verfügung (abhängig vom Datentyp des Modelleingabeparameters, auf den sie sich beziehen):

`EquidistantRangeParameterDeclaration` Ein Wert $w_n \in PW_i$ berechnet sich auf der Basis von drei Werten, die innerhalb des Sprachelementes angegeben werden: eines Startwertes w_0, eines Stopwertes w_{max} und einer konstanten Schrittweite s. Dabei gilt $w_n = w_{n-1} + s$, mit $w_n \leq w_{max}$.

`LogarithmicRangeParameterDeclaration` Hiermit kann eine logarithmische Annäherung an die obere Grenze definiert werden. Dadurch werden die Parameterwerte logarithmisch in der angegebenen Reichweite verteilt erzeugt, so dass an der oberen Grenze zunehmend mehr Parameterwerte liegen.

`SetIterationParameterDeclaration` Dieses Sprachelement stellt einen Aufzählungsdatentypen dar, in dem die Menge PW_i in Form von String-Literalen direkt angegeben wird (z. B. $PW_i = \{Literal_1, Literal_2\}$).

`BooleanIterationParameterDeclaration` Die Menge PW_i ist hierbei festgelegt auf die beiden Boolschen Werte: $PW_i = \{true, false\}$.

`VariationByExperimentPlan` Verweist auf einen zuvor im Workflow definierten Experimentplan und eine Ausgabeportdeklaration (`ObservableDeclaration`) einer dort verwendeten MUX-Deklaration. Dadurch lassen sich alle Modellausgabewerte (`ExperimentObservation`) aller im referenzierten Experimentplan gespeicherten Experimente[10] referenzieren, welche die Menge PW_i bilden.

Sprachelement `Experiment`

Der `ExperimentPlan` enthält auch die aus der Ausführungsspezifikation abgeleitete Deklarationen der Experimente (`Experiment`) mit den entsprechenden, konkreten Modelleingabeparameterwerten. Eine Experimentdeklaration e kann man auffassen als ein Element der Menge der kartesischen Produkte aller PW_i, so dass gilt: $e \in \times\, PW_i$. Diese Kreuzproduktmenge (und damit der zu betrachtende Parameterraum) kann beschränkt werden, indem das Sprachelement `ConstraintModel` verwendet wird (siehe folgendem Abschnitt 8.3.6).

Beispiel

Listing 8.2 zeigt einen Experimentplan für das Beispiel Bossel-Fischfang, in dem die maximale, spezifische Fangmenge Fisch pro Boot schrittweise erhöht wird (von 50 t bis 250 t mit

[10] Die Modellausgabewerte sind in den Experimentläufen gespeichert, wobei eine Referenz auf das Ergebnis eines Experimentes stets den zuletzt durchgeführten Experimentlauf meint.

Schrittweite 1, Zeile 6). Zudem werden zwei konstante Werte deklariert (ab Zeile 10). Für alle anderen Modelleingabeparameter gelten die Standardwerte, die in der MUX-Deklaration angegeben sind.

```
ExperimentPlan FISHFANG for FISHFANG.MOD {
 RangeMultiRunSpecification startAt "2013-03-31T11:00:00+02:00" parallel {
  executionEnvironment { /* dargestellt in Listing 8.4 auf Seite 117 */ }

  domainVariables { /* referenziert MUX-Deklaration in Listing 8.1 auf Seite 108 */
   EquidistantRange varyMaxSpezFangmenge {
    vary maxSpezFangmenge 50 -> 250 # 1 /* referenziert Listing 8.1, Zeile 18 */
   }
  }
  constants { /* referenziert ebenfalls die MUX-Deklaration in Listing 8.1 */
   FileConstantInputParameter constBoote { Boote = 25 }, /* referenziert Listing 8.1,
       Zeile 6 */
   FileConstantInputParameter constModellzeitstop { Modellzeitstop = 50 }
  }
 }
}

Legende: Schlüsselwort, Identifier, Referenz, "String", /* Kommentar */
```

Listing 8.2: Bossel-Fischfang-Beispiel für ExpL-Sprachelement `ExperimentPlan`, das die Variation der MUX-Eingabeparameterbelegung beschreibt wie in [Bos94, S. 175 f.]

8.3.4. Auswertungsplan

Das zweite deklarative Sprachelement in ExpL ist der Auswertungsplan (`EvaluationPlan`, siehe Abbildung 8.6 auf der nächsten Seite). Er beschreibt, wie die Modellausgabewerte nach Ausführung des MUX-Programmes weiter verwendet werden. Die kann z. B. die Berechnung einer Statistik auf der Basis der Ausgaben des MUX-Programmes in den Experimentläufen sein, die Erstellung eines Diagramms oder die GIS-DSL-basierte Auswertung in der SLEUTH-Fallstudie (Listing C.4 auf Seite 187).[11]

Hierbei wird im Auswertungsplan auch das Grundprinzip der „Trennung von Deklaration und Definition“ angewendet, so dass per `EvaluationDeclaration` angegeben wird, welche Modellausgaben genutzt werden sollen. Dies erfolgt per Referenz auf die `ObservableDeclaration` in der MUX-Deklaration und einen Experimentplan (mittels einer Assoziation zum `EvaluationPlan`, nicht dargestellt in Abbildung 8.6). Die durch den Auswertungsplan berechneten Ergebnisse werden im Sprachelement `EvaluationObservation` gespeichert (als Werte oder als Referenz auf ein Artefakt).

[11] Lediglich die GIS-DSL-basierte Auswertung in der SLEUTH-Fallstudie wurde als *proof of concept* implementiert. Es wäre wünschenswert, ExpL um allgemeine, statistische Methoden zu erweitern. Oftmals werden aber domänenspezifische Auswertungsmethoden benötigt, die als neue Aktivitäten entwickelt werden müssten.

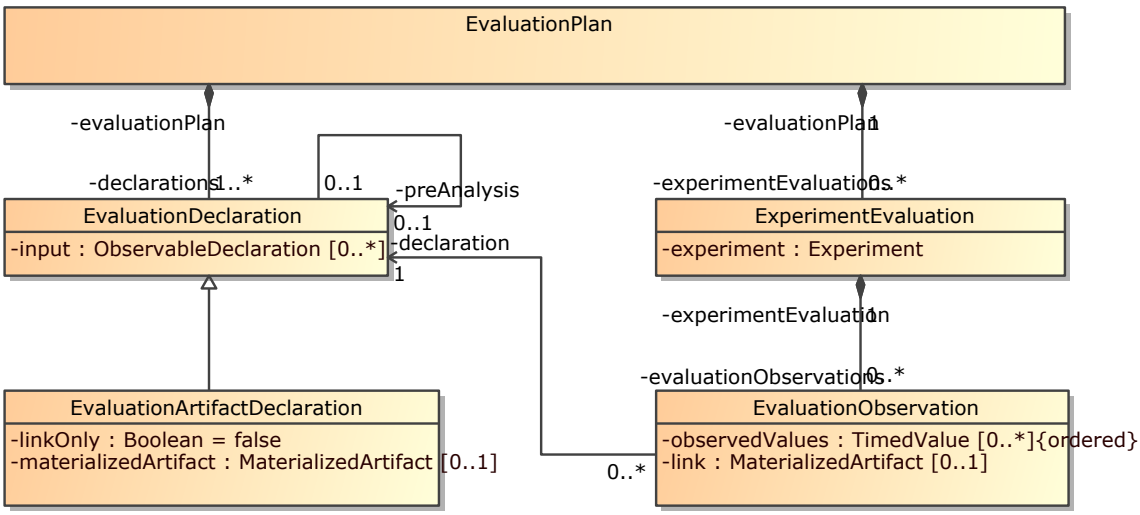


Abbildung 8.6.: Auszug des Klassendiagramms mit Sprachelement `EvaluationPlan` des ExpL-Sprachmetamodells (als UML-Klassendiagramm)

Eine weitere Analysemöglichkeit ist, innerhalb der `EvaluationDeclaration` auf einen zuvor in der ExpL-Wf-Definition gemachten Auswertungsplan zu verweisen (Schlüsselwort `preAnalysis`) und dessen berechnete Werte als Eingabe für einen neuen Auswertungsplan zu verwenden.

Beispiel

Listing 8.3 zeigt einen potentiellen Auswertungsplan für das Beispiel Bossel-Fischfang. Dabei wird angenommen, dass die Modellausgaben durch Ausführung des MUX-Programmes in der ExpL-Wf-Instanz gespeichert vorliegen (was z. B. in einer vorhergehenden Aktivität durch das Einlesen einer Datei erfolgt sein kann, die das MUX-Programm erzeugt hat – dargestellt in Zeile 1). Der Auswertungsplan (Zeile 4) ist fiktiv und das Sprachelement `TimelineDiagramm` (Zeile 6) ist nicht in ExpL implementiert. Es erzeugt ein Diagramm der Werteentwicklung von Fischen, Booten und Fangmengen über die Modellzeit (X-Achse, analog zu Abbildung 4.14 in [Bos94, S. 174 f.]).

8.3.5. Beschreibung der Ausführungsumgebung

Entsprechend des Grundprinzips der Artefakttrennung ist die Ausführungsumgebung eines MUX in ExpL separat beschrieben. Dies geschieht mittels des Sprachelementes `Execution Environment`, das Bestandteil einer Ausführungsspezifikation `(RunSpecification)` ist (siehe Klassendiagramm in Abbildung 8.7 auf Seite 116). Die Ausführungsumgebungsbeschreibung verfügt über eine Angabe, auf welchem Computer `(Machine)` das MUX-Programm ausgeführt werden soll. Prinzipiell lässt sich somit eine entfernte Ausführung auf einem anderen Computer umsetzen, als auf dem die steuernde ExpL-Workflow-Engine ausgeführt wird.

```
FishfangObservationFileReader FischfangDateileser { /* nicht implementiert; speichert
    Ausgaben des Fischfang-MUX-Programms in aktueller Workflow-Instanz */
  experimentPlan FISHFANG /* dargestellt in Listing 8.2 auf Seite 113 */ }
},
EvaluationPlan "FISHFANG Diagramm" {
  experimentPlan FISHFANG declarations {
    TimelineDiagramm "Boote Fische Fangmenge" { /* nicht implementiert */
      input (
        Boote, Fische, Fangmenge /* referenziert MUX-Deklaration in Listing 8.1, ab
            Zeile 24 */
      ),
      output (
        materializedArtifact diagram.png
      )
    }
  }
}

Legende: Schlüsselwort, Identifier, Referenz, "String", /* Kommentar */
```

Listing 8.3: Bossel-Fischfang-Beispiel für ExpL-Sprachelement `EvaluationPlan`, das ein Diagramm der Werteentwicklung von Fischen, Booten und Fangmengen über die Modellzeit erzeugt (wie in [Bos94, S. 174 f.])

Eine entfernte Ausführung erfordert in der aktuellen Implementierung hierfür jeweils eine eigene Instanz der ExpL-Wfe (wovon parallele Instanzen möglich sind, siehe Abschnitt 9.3.5).

Der zweite Bestandteil der Ausführungsumgebungsbeschreibung ist der `BinaryExecutorTask`. Bei dieser Aktivität wird davon ausgegangen, dass das MUX-Programm in Form eines passenden, ausführbaren Artefaktes vorliegt (`Executeable`) und keine Interaktion während der Ausführung zwischen ExpL-Wfe und MUX-Programm stattfindet. Modelleingabeparameterwerte und Laufzeitparameterwerte werden per Kommandozeilenparametern (`CommandLineParameter`) an das MUX-Programm übergeben. Der konkrete Wert des Kommandozeilenparameters wird durch eine Referenz auf eine Artefaktdeklaration bestimmt. Typischerweise ist das entweder eine Deklaration im Experimentplan oder eine Referenz des Standardwertes in der MUX-Deklaration. Es kann sich aber auch um Artefakte handeln, die von vorangegangenen Aktivitäten erzeugt wurden (z. B. eine Datei in einem MUX-programmspezifischen Eingabeformat).

Der `BinaryExecutorTask` startet das MUX-Programm zu der in der Ausführungsspezifikation angegebenen Zeit und wartet entweder auf dessen Terminierung oder auf den Ablauf des optionalen Watch-Dog-Timers (`watchDogTime`, Angabe in ms), der dann das MUX-Programm beendet. Während der Ausführung des MUX-Programms werden Monitoring-Informationen gesammelt. Es existieren verschiedene Spezialisierungen des `BinaryExecutorTask`, wie beispielsweise den `JavaExecutor`. Dieser erlaubt es, verschiedene Laufzeitparameterwerte an die Java-VM zu übergeben.

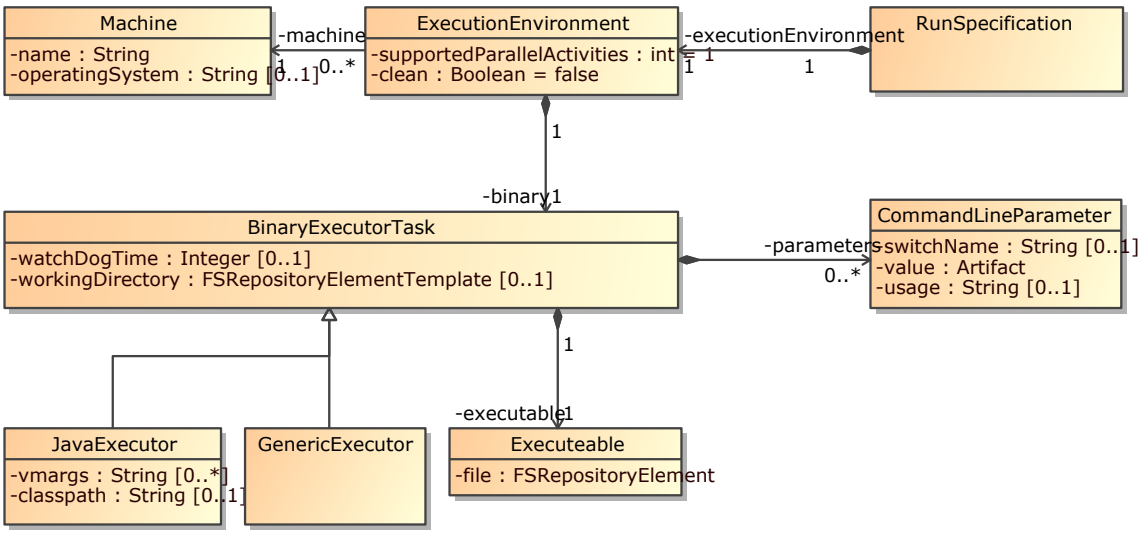


Abbildung 8.7.: Auszug des Klassendiagramms der Ausführungsumgebungsbeschreibung im ExpL-Sprachmetamodell (als UML-Klassendiagramm)

Beispiel

Listing 8.4 zeigt die verkürzte Beschreibung der Ausführungsumgebung für das Beispiel Bossel-Fischfang. Dabei wird von der Existenz des MUX-Programmes ausgegangen (Zeile 13), das beispielsweise durch eine vorangegangene Aktivität[12] im ExpL-Wf erzeugt wurde. In [Bos94, S. 143 f.] ist nur ein interaktiver, GUI-basierter Modus zur Ausführung des MUX-Programmes beschrieben. In diesem Beispiel wird aber davon ausgegangen, dass es sich um eine nichtinteraktive Version handelt, welche die Modelleingabeparameterwerte per Kommandozeile übergeben bekommt. Pro Deklaration eines Modelleingabeparameters in der MUX-Deklaration wird genau ein Kommandozeilenparameter gleichen (kleingeschriebenen) Namens angenommen.

8.3.6. Beschränkung des Parameterraums

In Abschnitt 8.3.3 wurde beschrieben, wie der zu betrachtende Parameterraum durch die Kreuzproduktmenge $\times PW_i$ gebildet werden kann. Diese Kreuzproduktmenge kann sehr groß werden, und die daraus resultierenden Experimente können aufgrund ihrer Anzahl oftmals nicht praktisch durchgeführt werden (z. B. aus Zeitgründen). Insbesondere kann die Kreuzproduktmenge unerwünschte Parameterwertekombinationen enthalten, die für das Untersuchungsziel unnötig sind.

Deshalb bietet ExpL das Sprachelement `ConstraintModel`, mit dem sich die initiale Parameterwertebelegung des MUX als *Constraint-Satisfaction-Problem (CSP)* formulieren lässt. Dabei stehen boolsche und algebraische Ausdrücke zur Verfügung, die über Konstanten (`Con-`

[12] Eine solche Aktivität ist z. B. ein `AntScriptTask`, mit dem ein Build-Skript im Werkzeug Ant [ant13] ausgeführt werden kann.

```
ExperimentPlan FISHFANG for FISHFANG.MOD {
 RangeMultiRunSpecification startAt "2013-03-31T11:00:00+02:00" parallel {
  parallel executionEnvironment clean {
   executionEnvironment clean {
    supportedParallelActivities 5 binary GenericExecutor FISHFANG {
     workingDirectory wd_exp_id parameters {
       "--fangmenge" = varyMaxSpezFangmenge, /* referenziert ExperimentPlan in
           Listing 8.2, Zeile 6 */
       "--boote" = constBoote, /* referenziert Konstanten aus ExperimentPlan in
           Listing 8.2, ab Zeile 10 */
       "--modellzeitstop" = constModellzeitstop,
       "--fanggebiet" = Fanggebiet, /* referenziert MUX-Deklaration in Listing 8.1,
           Zeile 11 */
       /* ... */
     }
     executable Executeable { file SIMPAS.EXE }
    }
    machine experimentserver.informatik.hu-berlin.de
   }
   /* ... */
  }
}

Legende: Schlüsselwort, Identifier, Referenz, "String", /* Kommentar */
```

Listing 8.4: Bossel-Fischfang-Beispiel für ExpL-Sprachelement `ExecutionEnvironment`, das die Ausführung des MUX-Programmes beschreibt wie in [Bos94, S. 143 f.]

`stantsDeclaration`) und Variablen (`ParameterVariationDeclaration`) des Experimentplanes oder temporären Variablen gebildet werden können (siehe Klassendiagramm im Anhang in Abbildung A.11 auf Seite 174). Durch Verweise auf Konstanten und Variablen des Experimentplanes können die bereits dort definierten Eigenschaften übernommen werden, und die Referenzen auf die entsprechenden Modelleingabeparameter in der MUX-Deklaration sind dadurch implizit bekannt (wodurch die Lösung des formulierten CSP zur Bildung der Experimente nutzbar ist). Ein Beispiel findet sich im NetTopo-Beispiel im Anhang B.3 (Listing B.4, ab Zeile 44).

Das CSP wird mittels Constraintprogrammierung bearbeitet, wobei auf das Standardformat MiniZink abgebildet wird, das von dem existierenden Werkzeug JaCoP [KS11] verwendet wird. Durch JaCoP ergibt sich die Einschränkung, dass nur Integer als Datentyp für Variablen in algebraischen Ausdrücken zugelassen ist. In algebraischen Ausdrücken werden die Operationen Addition (Subtraktion mittels unärem Minus-Operator), Multiplikation und Division unterstützt. Als boolsche Operationen werden angeboten: nicht, gleich, ungleich, kleiner, kleiner gleich, oder, und, exklusives oder.

8.3.7. Minimalbeispiel

Listing 8.5 zeigt die textuelle Repräsentation einer minimalen ExpL-Wf-Definition mit den vorgestellten Sprachelementen, beginnend bei Ressourcen (`Machine`, Zeile 6 und Repositorien, Zeile 8) und Aktivitäten (Zeile 27). Insbesondere die ExpL-Wf-Definition wird als Artefakt referenziert (Zeilen 3 und 12).

```
1  ExperimentationWorkflow {
2    description "Gerüst einer ExpL-Wf-Definition"
3    resource = stub /* ExpL-Wf-Definition, referenziert Deklaration in Zeile 12 */
4    resultRepository = modellRepository /* Referenziert Deklaration in Zeile 15 */
5    resources {
6      Machine "experimentserver.informatik.hu-berlin.de" { }
7    }
8    repositories {
9      CDORepository experimentRepositorium {
10       ConfigurationRepository URI = "tcp://experimentserver.informatik.hu-berlin.de:2036"
11       elements {
12         CDORepositoryElement stub { }
13       }
14     },
15     SVNRepository modellRepository {
16       ModelRepository URI =
17           "svn+ssh://experimentserver.informatik.hu-berlin.de/repo/project1/models"
18       workingDirectory = modellsLocalCache /* Referenziert Deklaration in Zeile 22 */
19       elements {
20         SVNModelRepositoryElement MUX { modelName MUX }
21       }
22     },
23     FSRepository modellsLocalCache {
24       ModelRepository URI = "file:///experiments/test"
25       elements { FSRepositoryElement MUX_Programm }
26     }
27   }
28   tasks {
29     ExperimentPlan "Variation der Eingabeparameter" for MUX { /* Referenziert
           Deklaration in Zeile 19 */
30       SingleRun {
31         executionEnvironment {
32           binary JavaExecutor "Ausführbares Programm des MUX" {
33             executable Executeable { file MUX_Programm } /* Referenziert Deklaration in
                   Zeile 24 */
34           }
35           machine experimentserver.informatik.hu-berlin.de /* Referenziert Ressource in
                 Zeile 6 */
36 } } } } }
37
38 Legende: Schlüsselwort, Identifier, Referenz, "String", /* Kommentar */
```

Listing 8.5: Minimale Exp-Wf-Definition in der Notationssprache von ExpL

9. Architektur des ExpL-Workflow-Systems

In Kapitel 8 wurde die Semantik der wesentlichen Bestandteile des ExpL-Sprachmetamodells vorgestellt und anhand von Beispielen für ExpL-Wf-Definitionen illustriert. Das ExpL-Sprachmetamodell, die Notationssprache von ExpL, die ExpL-Wf-Definitionen und -Instanzen, sowie das ExpL-Workflow-Management-System bilden zusammen das ExpL-Workflow-System (siehe Abbildung 9.1). In diesem Kapitel 9 werden die Architektur, die Bestandteile und die technische Umsetzung des ExpL-Workflow-Systems dargestellt.

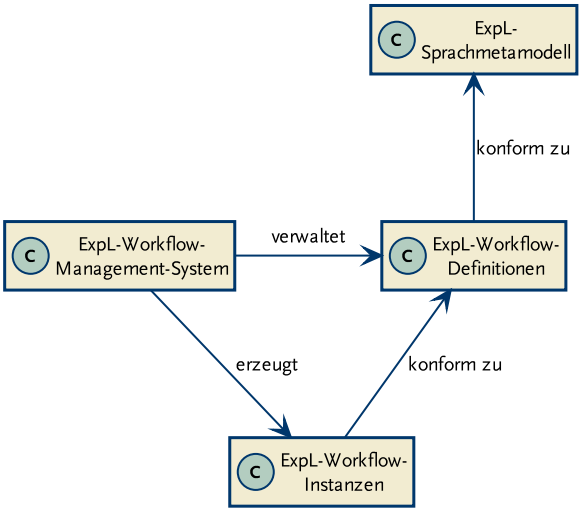


Abbildung 9.1.: Bestandteile des ExpL-Wf-Systems (als UML-Klassendiagramm; erweitert Abbildung 8.1 auf Seite 102)

9.1. Basistechnologien

9.1.1. Modellgetriebene Softwareentwicklung

Das ExpL-Workflow-System wurde mit Hilfe von Techniken der modellgetriebenen Softwareentwicklung umgesetzt, auf die an dieser Stelle überblicksartig eingegangen wird. Dabei ist nach [SV07, S. 11] die modellgetriebene Softwareentwicklung – *Model-Driven-Software-Development (MDSD)*[1] – „ein Oberbegriff für Techniken, die aus formalen Modellen automatisiert lauffähige Software erzeugen“. Unter diesen Softwarebegriff fällt allgemein nicht

[1] Für MDSD wird oftmals das Synonym *Model-Driven-Development (MDD)* verwendet.

nur Quelltext, der durch einen Compiler in Maschinencode übersetzt wird, so dass dieser auf einem Computer ausgeführt werden kann, sondern er schließt auch andere Arten von Artefakten ein, welche für die Lauffähigkeit der Software notwendig sind (z. B. Konfigurationsdateien, Datenbankschemata oder auch Dokumentation). Im ExpL-Workflow-System wird insbesondere die Erzeugung von Artefakten und Sprachwerkzeugen (für ExpL) mit Techniken der modellgetriebenen Softwareentwicklung umgesetzt.

In MDSD sind formale Modelle der Ausgangspunkt, die Softwaresysteme in Struktur, Verhalten und Umgebung beschreiben. Dabei existieren die zu beschreibenden Softwaresysteme noch nicht real. Vereinfachend ausgedrückt, werden zunächst die Anforderungen an ein Softwaresystem informal und anschließend formal spezifiziert.[2] Darauf aufbauend, wird ein formales Modell erstellt, auf dessen Basis automatisiert das Softwaresystem (bzw. Teile davon) erzeugt wird.

Eine prominente Umsetzung der Ideen von MDSD liefert die OMG[3] mit dem Standard *Model-Driven-Architecture (MDA)* [MM03]. Darin wird das formale Ausgangsmodell zur Erstellung eines Softwaresystems als *Platform Independent Model (PIM)* bezeichnet. Der Name weist bereits darauf hin, dass implementierungsspezifische Aspekte, wie beispielsweise eine bestimmte Zielsprache (z. B. eine GPL) nicht Bestandteil des PIM sein sollen.

Zentral in MDSD, und somit auch in MDA, sind sogenannte *Transformationen*, mit denen aus entsprechenden Modellen Softwareartefakte erzeugt werden. Die Philosophie von MDA sieht vor, alle implementierungsspezifischen Aspekte einer Zielplattform in einer (plattformspezifischen) Transformation auszudrücken, die somit aus einem Platform Independent Model ein *Platform Specific Model (PSM)* erstellen kann.[4] Ein solcher Prozess wird allgemein *Transformationsprozess* genannt, den eine bestimmte Software ausführt, die *Generator* genannt wird. Das PSM vereint somit plattformunabhängige und plattformspezifische Aspekte. Im ExpL-Workflow-System wird das ExpL-Sprachmetamodell als PIM verstanden, aus dem verschiedene PSM transformiert werden (z. B. Sprachwerkzeuge und Konfigurationsdateien für verwendete Werkzeuge und Bibliotheken).

Dabei werden im Kontext von modellgetriebener Softwareentwicklung zwei Arten von unidirektionalen Transformationen unterschieden, die im Folgenden wichtig sind: *Modell-zu-Modell (M2M)* und *Modell-zu-Code (M2C)* [CH03]. M2M-Transformationen erzeugen aus einem formalen Modell ein neues, formales Modell, wobei sich die verwendeten Formalismen unterscheiden können (z. B. verschiedene Metamodelle). Eine M2M-Transformation, bei der kein neues Modell erzeugt wird, sondern das vorhandene abgeändert wird, bezeichnet

[2] Zu Techniken der Softwareentwicklung existiert eine Fülle an Literatur: Exemplarisch sei bezüglich allgemeiner Softwareentwicklung auf die Bücher von Balzert verwiesen [Bal09, Bal11] und speziell zu MDSD wird z. B. auf Stahl verwiesen [SV07].

[3] Die *Object Management Group (OMG)* ist ein 1989 gegründetes, internationales Konsortium der Computerindustrie (und auch Endnutzern von Software), das nicht gewinnorientiert arbeitet. Die OMG erarbeitet Spezifikationen und Standards, um Interoperabilität und Portabilität von Softwaresystemen zu verbessern.

[4] Obwohl PIM und PSM spezifische Begriffe der OMG im Kontext von MDA sind, werden beide Begriffe in dieser Arbeit hiervon losgelöst in ihrem konzeptionellen Sinn benutzt, um den Unterschied zwischen plattformunabhängigen und plattformspezifischen Aspekten in MDSD zu bezeichnen.

man als *Modellmodifikation*. M2C-Transformationen erzeugen aus einem formalen Modell Text in einer Computersprache (Quellcode). Eine weiter gefasste Bezeichnung dieser Transformationsart ist daher *Modell-zu-Text (M2T)*, wobei der erzeugte Text nicht mehr zwingend maschinenlesbar sein muss (z. B. ein Bericht für einen Menschen).

Ein weiterer Begriff im Kontext von modellbasierter Softwareentwicklung ist MDE [Sch06b]. In der Literatur wird der Begriff MDE nicht einheitlich verwendet. Oftmals findet man ihn synonym zu MDSD. Andernorts herrscht ein gewisser Konsens, dass MDE weiter gefasst ist als MDSD und beispielsweise auch Techniken wie modellbasiertes Testen und modellgetriebenes Reverse-Engineering einschließt [Sch11, S. 19]. Abbildung 9.2 visualisiert die Zusammenhänge (auf EMF wird im Folgenden Abschnitt 9.1.2 eingegangen). Der MDA-Standard schreibt die Verwendung von UML vor (u. a. zur Beschreibung des PIM), wohingegen MDSD an keine spezifische Technologie gebunden ist.

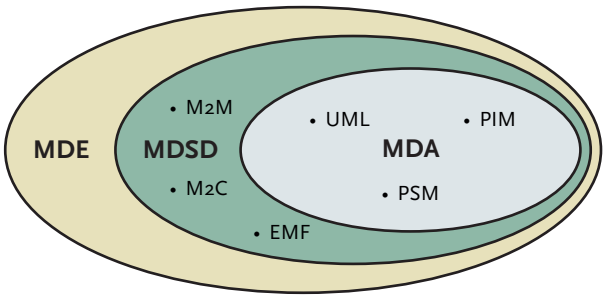


Abbildung 9.2.: Relationen zwischen MDE ↔ MDSD ↔ MDA und zugehörige Technologien

9.1.2. Ecore-Metametamodell und das Eclipse Modeling Framework

Im Modelware-Kontext werden Metamodelle, wie das ExpL-Sprachmetamodell, in Form von Klassendiagrammen notiert. Als Notationssprachen hierfür haben sich die UML(-Klassendiagramme) und *Ecore* durchgesetzt. Ecore ist das *Metametamodell* im *Eclipse Modeling Framework (EMF)* [SBPM09]. Das ExpL-Sprachmetamodell liegt als Modell des Ecore-Metametamodells (verkürzt *Ecore-Modell*) vor.[5] Ecore ist eine Implementierung der *Essential Meta Object Facility (EMOF)*, einer Teilmenge der *Complete Meta Object Facility (CMOF)* [20006a]. CMOF seinerseits ist eine spezielle Teilmenge von Konzepten, die in UML-Klassendiagrammen benutzt werden. Die OMG verwendet CMOF *rekursiv* zur Definition von UML 2.x (von Klassendiagrammen und allen anderen Diagrammarten).

[5] Zu Beginn der Arbeiten am ExpL-Sprachmetamodell waren UML-Editoren weiter entwickelt, als ihre Gegenstücke im EMF. Deshalb wurde ein solcher Editor verwendet, um das ExpL-Sprachmetamodell in Form von UML-Klassendiagrammen zu notieren, welche für die weitere Verwendung mittels Standardtechnologien in ein Modell des Ecore-Metametamodells (verkürzt *Ecore-Modell*) transformiert werden. Inzwischen wäre eine direkte Erstellung als Ecore-Modell ähnlich komfortabel.

Der Eclipse-Technologieraum für MDSD heißt *Eclipse Modeling Project* und umfasst nicht nur das EMF-Projekt, sondern auch die Projekte *Textual Modeling Framework (TMF)* und *Model to Text Transformation* (M2T), welche Technologien beinhalten, die im ExpL-Workflow-System eingesetzt werden (siehe Abbildung 9.3). Die Bezeichnung EMF wird sowohl für den Namen eines Projektes der *Eclipse Foundation* verwendet, als auch für das dort entwickelte Modellierungs- und Code-Generierungsframework (auf der Basis von Ecore). In dieser Arbeit wird *EMF* im Sinne des Frameworks benutzt und *EMF-Projekt* für die entsprechende Projektinitiative.

Nach Meinung des Autors dieser Arbeit stellt der Eclipse-Technologieraum gegenwärtig die fortschrittlichsten Werkzeuge für eine praktikable Anwendung[6] von MDSD bereit, weshalb er für die Implementierung des ExpL-Workflow-Systems gewählt wurde. Insbesondere das Vorhandensein eines Modellrepositoriums (CDO, siehe Abschnitt 9.1.4), welches einen transaktionalen, verteilten Zugriff erlaubt, war für die Wahl des Eclipse-Technologieraumes entscheidend. Zudem sind alle Werkzeuge aus dem Eclipse-Technologieraum, die für die Implementierung des ExpL-Workflow-Systems gewählt wurden, quelloffen und frei, nicht-kommerziell nutzbar.

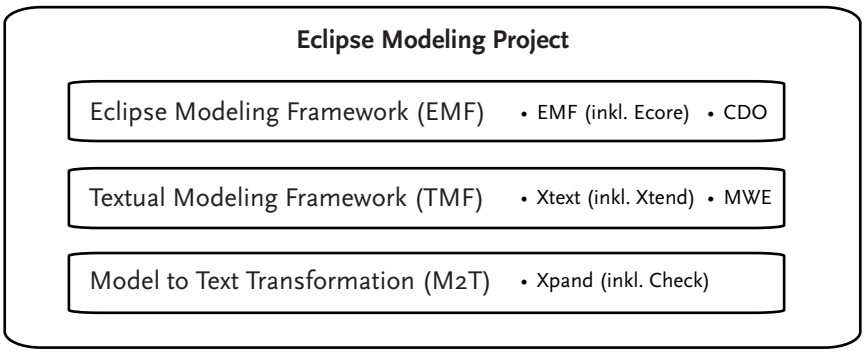


Abbildung 9.3.: Der Eclipse-Technologieraum für MDSD im *Eclipse Modeling Project* mit den zugehörigen Unterprojekten und Technologien, die im ExpL-Workflow-System genutzt werden

9.1.3. Xtext, Xpand, Xtend und Check

Bis Ende 2008 waren die Werkzeuge *Xtext*, *Xpand*, *Xtend*, *Check* und MWE Bestandteile des Frameworks *openArchitectureWare (oAW)*, entwickelt von der deutschen Firma *itemis AG*. Seit 2009 werden sie im Rahmen des *Eclipse Modeling Project* in unterschiedlichen Unterprojekten weiter entwickelt. Um die Zugehörigkeit zu Eclipse zu kennzeichnen, werden stellenweise die erweiterten Bezeichnungen Xtext2, M2T Xpand und MWE2 benutzt. Da dies nicht einheitlich geschieht, werden in dieser Arbeit die Bezeichnungen ohne Erweiterung verwendet.

[6] Auf einen Vergleich des Eclipse-Technologieraums mit den MDA-Technologien der OMG muss an dieser Stelle verzichtet werden. Einige Aspekte im Vergleich zwischen UML und Ecore liefert jedoch z. B. Scheidgen [Sch09, S. 39 ff.].

Xtext

Xtext gehört zum Textual Modeling Framework und ermöglicht die Erstellung von DSLs [EKF08]. Dabei kann mit Xtext sowohl die Syntax einer konkreten Sprachinstanz (in Textform), als auch die Syntax einer abstrakten Sprachinstanz (in Form eines Ecore-konformen Modells) beschrieben werden. Xtext bietet selbst eine grammatikbasierte Beschreibungssprache, die zur Definition der Notationssprache von ExpL benutzt wird.[7] Die Notationssprache von ExpL stellt dabei eine konkreten Sprachinstanz in Textform dar. Die Beschreibungssprache von Xtext erlaubt das Erstellen von Regeln, die aus Zeichenketten und Referenzen auf Elemente des ExpL-Sprachmetamodells bestehen. Das ExpL-Sprachmetamodell ist hierbei bereits eine Syntax einer abstrakten Sprachinstanz und muss nicht von Xtext erzeugt werden. Xtext generiert nicht nur einen Parser für textuelle Modelle der Notationssprache von ExpL, sondern bietet auch die Generierung eines passenden, in Eclipse integrierten Texteditors und einer Infrastruktur für die Implementierung der zugehörigen Semantik an.

Zu Xtext existieren im Eclipse-Technologieraum die Alternativen *Textual Editing Framework (TEF)* [Sch10] und EMFText. Schmidt hat in [Sch11] einen Vergleich von Xtext, TEF und EMFText vorgenommen. Anhand der drei Kriterien *Aufwandsabschätzung bei der Einarbeitung in das Framework*, *Lebensdauer* und *Dokumentation und Support* ist Xtext bei diesem Vergleich den genannten Alternativen deutlich überlegen.

Xpand

Xpand ist eine statisch getypte Templatesprache, die zur Definition der M2C-Transformationen im ExpL-WfMS eingesetzt wird [Xpa13]. Der Name „Xpand“ bezieht sich auf das englisch gleichlautende *expand*, zu deutsch „ausweiten“ (oder „expandieren“). Damit wird das Grundprinzip der Templatesprache beschrieben: Anhand von Transformationsregeln lassen sich die Modellelemente des Eingabemodells der Transformation entlang ihrer Relationen durchlaufen, die in dem zugehörigen Metamodell definiert sind. Dabei kann auf die Attributwerte und Typinformationen der Modellelemente zugegriffen werden. Diese können um Zeichenketten erweitert werden und bilden als Code das Ergebnis der Transformation. Xpand unterstützt zudem den polymorphen Aufruf und die Modularisierung von Templates. Im Gegensatz zu Xtext werden durch Xpand weder ein Parser und noch ein Editor für den durch die M2C-Transformation erstellten Code angeboten.

Check

Check ist ein Teil von Xpand und erlaubt es, Überprüfungen (*checks*) von Modellelementen und deren Relationen vorzunehmen. Eine solche Überprüfung wird *Modell-Checking* genannt. Check ist äquivalent zur *Object Constraint Language (OCL)* der OMG, arbeitet jedoch auf Basis des Ecore-Metametamodells. Metamodelle, die konform zum Ecore-Meta-

[7] Xtext basiert auf *ANother Tool for Language Recognition (ANTLR)*, einem objektorientiert arbeitendem Parsergenerator, der LL(k)-Grammatiken mit beliebigen k verarbeiten kann [Par07].

metamodell sind, definieren nur die Struktur von Elementen und deren strukturelle Beziehungen untereinander. Eine Instanz (bzw. ein konformes Modell) eines solchen Metamodells besteht aus konkreten Ausprägungen dieser Beziehungen (z. B. kann eine n:m-Relation als 2:3-Relation instanziiert werden). Modell-Checking erlaubt es, diese konkreten Ausprägungen durch Regeln zu überprüfen, die sonst nicht in einem (Ecore-konformen) Metamodell ausgedrückt werden können. Anhand dieser Regeln kann ein Modell als nicht konform abgelehnt werden.

Jede Überprüfungsregel in Check referenziert mittels des Schlüsselwortes `context` ein Metamodellelement, das zu der entsprechenden Instanz des zu überprüfenden Modellelementes passt. Dieses Metamodellelement besitzt Relationen und Attribute, die in den Ausdrücken der Überprüfungsregel ausgewertet werden können. Eine fehlgeschlagene Überprüfung kann entweder einen Fehler oder eine Warnung auslösen und mit einer Textmeldung versehen werden, in der Werte des Modells ausgegeben werden können. Eine Überprüfungsregel hat folgenden, schematischen Aufbau:

```
context Metamodellelement ERROR "Nachricht": Ausdruck; //wahr/falsch
```

Check wird eingesetzt, um die Einhaltung semantischer Beschränkungen für ExpL-Wf-Definitionen zu überprüfen, wie beispielsweise, ob ein Datei-Artefakt zugreifbar ist (siehe Listing D.2 auf Seite 200 im Anhang). Mit Check ließe sich auch überprüfen, ob Wertzuweisungen kompatibel hinsichtlich ihrer zugewiesenen Einheitentypen sind.

Xtend

Innerhalb des Projektes TMF ist Xtend zu einer umfangreichen und ausdrucksstarken funktionalen Sprache auf der Basis von Java geworden [EZ11]. Xtend erlaubt es, Modellelemente um zusätzliche Logik zu erweitern (sogenannte *Extensions*, wie sie auch in der Sprache C# 3.0 vorkommen). Xtend ist vergleichbar mit Sprachen wie Scala und Groovy.

Das ExpL-WfMS setzt Xtend als Bindeglied von Xpand und Check zu Java ein (wie es auch seine ursprüngliche Funktion in oAW war). Dies bedeutet beispielsweise, Funktionen für das Expandieren von Metamodellelementen werden in Java beschrieben und mittels Xtend registriert, um sie dann in den Transformationsregeln von Xpand zu verwenden. Eine Deklaration in Xtend zum Aufruf von Javacode sieht beispielsweise wie folgt aus:

```
String getCurrentISO8601Date(): JAVA ISO8601DateParser.getCurrentIsoDate();
```

9.1.4. Connected Data Objects

Mit *Connected Data Objects (CDO)* lassen sich Ecore-basierte Modelle persistent speichern und zugreifbar machen [CDO13]. CDO ist ein Softwaresystem, das auf EMF basiert und eine Client-Server-Architektur umsetzt. Der Server nutzt dabei zur Persistierung ein *relationales Datenbankmanagementsystem (RDBMS)*, wie z. B. Derby[8], HSQLDB oder MySQL. Eine bi-

[8] *Derby* ist ein Java-basiertes, relationales Datenbankmanagementsystem, das im Projekt *Apache Derby* der *Apache Software Foundation* entwickelt wird. Derby wird innerhalb von CDO im ExpL-WfMS eingesetzt.

direktionale, objektrelationale Abbildung ermöglicht das Speichern und das Laden der Ecore-basierten Modelle. CDO setzt das AKID-Paradigma um, wobei es ein Transaktionskonzept verwendet. Dadurch können mehrere CDO-Clients gleichzeitig und voneinander isoliert auf den Ecore-basierten Modellen arbeiten.

Mit CDO ist im ExpL-WfMS das Experiment-Repositorium umgesetzt worden, das ExpL-Wf-Definitionen und -Instanzen speichert. Hierbei ist das Experiment-Repositorium ein CDO-Server, an dem das ExpL-Sprachmetamodell registriert wurde. Das ExpL-WfMS hat zwei Bestandteile, die CDO-Clients sind: die ExpL-Wfe und den ExpL-Baumeditor (ein ExpL-Workflow-Editor mit dem ExpL-Wf-Definitionen erstellt werden können).[9]

9.1.5. Modeling Workflow Engine

Die *Modeling Workflow Engine (MWE)* ist ein einfaches, Java-basiertes Workflow-System, das aus einer textuellen Beschreibungssprache für Workflows, einem Texteditor, der diese Sprache unterstützt, und einer Workflow-Engine besteht [20113a]. Sowohl die MWE-Beschreibungssprache als auch der hierfür passende Texteditor wurden mit Xtext erstellt. Xtext wiederum verwendet MWE zur Steuerung von Generierungsaufgaben (z. B. zur Erzeugung von Parser und Texteditor). MWE definiert nur Kontrollflüsse zwischen Aktivitäten, die in MWE *Components* heißen. Datenflüsse zwischen Aktivitäten werden nicht beschrieben, weil alle Aktivitäten (eines Workflows) auf einen gemeinsam genutzten Speicher (*shared memory*) zugreifen können. MWE unterstützt nur sequentielle Abläufe von Aktivitäten. In ExpL sind zusätzlich parallele Abläufe und (implizite) Iterationen, sowie Verschachtelungen von sequentiellen und parallelen Abläufen von Aktivitäten möglich. Um im ExpL-WfMS eingesetzt werden zu können, wurde MWE um Verschachtelungen und parallele Abläufe erweitert (Iterationen werden auf sequentielle Abläufe abgebildet).

MWE bietet eine Reihe vordefinierter Aktivitäten (z. B. zum Laden und Speichern von Ecore-Modellen), die jedoch nicht ausreichend für die im ExpL-WfMS benötigten Funktionen waren. Es fehlten beispielsweise Aktivitäten zur Interaktion mit CDO: Deshalb wurden Aktivitäten zum Laden und Speichern von ExpL-Wf-Definitionen, sowie zur Nutzung von Transaktionen in MWE implementiert.

9.2. Grundprinzipien

Die prototypische Implementierung des ExpL-WfMS basiert auf einigen Grundprinzipien bzw. Designentscheidungen, die im Folgenden erläutert werden.

[9] Der ExpL-Texteditor ist bisher technisch gesehen nicht als CDO-Client umgesetzt, was eine Verbesserung der Architektur wäre. Gegenwärtig übernimmt die ExpL-Wfe das Laden und Speichern der ExpL-Wf-Definitionen, die mit dem ExpL-Texteditor erzeugt wurden.

Gemeinsam genutzter Speicher

In ExpL sind Datenflüsse nicht explizit beschreibbar, wie es etwa durch die Spezifikation von Eingabe- und Ausgabeschnittstellen an Aktivitäten umsetzbar wäre. Alle Aktivitäten in ExpL können auf einen gemeinsam genutzten Speicher in der ExpL-Wf-Instanz zugreifen, der in Form eines zum ExpL-Sprachmetamodell konformen Modells vorliegt. Somit sind die möglichen Strukturen, die von Aktivitäten hinzugefügt oder verändert werden können, durch das ExpL-Sprachmetamodell vorgegeben. Dadurch stehen zugleich umfangreiche Möglichkeiten zur Strukturierung dieser Daten zur Verfügung. Ein typisches Beispiel für die Nutzung des gemeinsamen Speichers ist die Speicherung von Modellbeobachtungen innerhalb des ExpL-Sprachelementes `Experiment`.

Der gemeinsam genutzte Speicher wird durch die ExpL-Wfe verwaltet. In der aktuellen Implementierung besteht kein Zugriffsschutz zwischen Aktivitäten, was bedeutet, dass jede Aktivität auch die Daten einer anderen Aktivität verändern könnte. Typischerweise wird der gemeinsam genutzte Speicher allerdings lesend verwendet, um auf Daten der ExpL-Wf-Definition zuzugreifen, beispielsweise, um eine MUX-Deklaration auszulesen. Ein Schreibschutz für die Daten der ExpL-Wf-Definition wäre wiederum wünschenswert, ist aber in der aktuellen Implementierung nicht vorhanden. Der gemeinsam genutzte Speicher wird durch die ExpL-Wfe mittels CDO permanent gesichert. Für Daten, die nur temporär während der Ausführung der ExpL-Wf-Instanz zwischen Aktivitätsinstanzen weitergegeben werden müssen, steht ein nichtpermanenter Speicher innerhalb von MWE zur Verfügung, der ebenfalls dem Konzept eines gemeinsam (durch MWE-Aktivitäten) genutzten Speichers folgt.

Ressourcen-Zuweisung und Zeitablaufsteuerung zur Entwurfszeit

In ExpL werden alle durch Aktivitäten benötigte Ressourcen zur Entwurfszeit der ExpL-Wf-Definition an die verfügbaren Ressourcen gebunden. Beispielsweise wird innerhalb einer Ausführungsumgebungsbeschreibung auf eine konkrete Maschine verwiesen. Das ExpL-WfMS führt generell keine Analysen der auszuführenden ExpL-Wf-Definition bzw. -Instanz durch, um eine bestimmte Ausführungsreihenfolge der Aktivitätsinstanzen der ExpL-Wf-Instanz zu erreichen. Der Ablauf einer ExpL-Wf-Instanz unterscheidet sich nicht von einer anderen, der die selbe ExpL-Wf-Definition zugrunde liegt.

ExpL könnte jedoch um Sprachelemente erweitert werden, die komplexere Ausführungsumgebungen, wie beispielsweise Computing-Cluster unterstützen. So könnte innerhalb einer Ausführungsumgebungsbeschreibung auf eine Ressource verwiesen werden, die einen Pool von Maschinen eines Computing-Clusters darstellt. Die restliche ExpL-Wf-Definition bliebe dabei unverändert. Das ExpL-WfMS hätte dann zur Laufzeit der ExpL-Wf-Instanz zu entscheiden, welche Maschine des Pools genutzt wird.

ExpL-Workflows werden komplett zu Zeitpunkten ausgeführt, die zur Entwurfszeit festgelegten werden. Eine flexiblere Zeitablaufsteuerung auf Ebene der Aktivitäten oder auch durch ein externes Managementsystem wäre allerdings, gerade im Hinblick auf Computing-Cluster, eine sinnvolle Erweiterung (dieses Szenario wird in Abschnitt 13 weiter diskutiert).

Unvollständige ExpL-Wf-Definitionen

Eine ExpL-Wf-Definition muss konform zum ExpL-Sprachmetamodell sein, um gespeichert, instanziiert und ausgeführt werden zu können. Während des Editierens einer ExpL-Wf-Definition sind oftmals noch nicht alle notwendigen Elemente eingegeben worden, um konform zum ExpL-Sprachmetamodell zu sein. Solche unvollständigen Modelle lassen sich normalerweise nicht weiter durch Metamodellierungswerkzeuge verarbeiten. Insbesondere kann eine unvollständige ExpL-Wf-Definition nicht im Experiment-Repositorium durch CDO gespeichert werden.

ExpL-Wf-Definition können sehr umfangreich werden. Es wäre benutzerunfreundlich zu verlangen, sie erst speichern zu können, wenn sie vollständig eingegeben wurde. Aus diesem Grunde behilft man sich in der Metamodellierung mit einem Trick, um auch unvollständige Modelle zuzulassen. Das Metamodell wird strukturell weniger restriktiv formuliert, als es für das zu beschreibende System nötig wäre. In ExpL wird dieser Trick angewendet, wodurch sich unvollständige Workflows definieren lassen (z B. ein Experimentplan ohne Ausführungsumgebungsbeschreibung). Weil sich ein solch unvollständiger Workflow nicht ausführen lässt, muss vor bzw. bei dessen Instanziierung überprüft werden, ob er zu diesem Zeitpunkt vollständig definiert ist. In ExpL geschieht dies mittels Modell-Checking durch die ExpL-Wfe.

Provenance

Das ExpL-WfMS bietet Unterstützung für Provenance durch die Aufzeichnung von Veränderungen, die an einer ExpL-Wf-Instanz vorgenommen werden. Jede Aktivitätsinstanz speichert dabei nach ihrer Abarbeitung eine neue, potentiell veränderte Version der ihr zugrundeliegenden ExpL-Wf-Instanz in CDO. Die Änderungen, welche diese bestimmte Aktivitätsinstanz vorgenommen hat, lassen sich somit als Differenz der Versionen vor und nach ihrer Abarbeitung bestimmen. Andere Änderungen, die möglicherweise an externen, nicht der Kontrolle des ExpL-WfMS unterliegenden Artefakten vorgenommen wurden, lassen sich nicht automatisch durch das ExpL-WfMS nachvollziehen. Allerdings könnte eine Aktivität selbst für Provenance-Daten sorgen, beispielsweise, indem es eine neue Version eines externen Artefaktes anlegt, das als Ressource mit einem PI dem ExpL-WfMS bekannt gemacht wird.

Mit dem Aufzeichnen und Auswerten solcher Daten im metamodellbasierten ExpL-WfMS beschäftigt sich die Diplomarbeit von Yu [Yu12]. Dabei entstand eine Erweiterung für das ExpL-WfMS, mit der Differenzen zweier ExpL-Wf-Instanzversionen visualisiert und ausgewertet werden können. Die Auswertung erfolgt mittels einer einfachen DSL, welche die OCL um temporallogische Ausdrücke erweitert. Damit können Anfragen auf Experimentdaten, wie Modellbeobachtungen und Monitoring-Informationen, gestellt werden (z. B. ob jedes der Experimente in einer bestimmten Zeitdauer ausgeführt wurde).

9.3. ExpL-Workflow-Management-System

Die Bezeichnung ExpL-*Workflow-Management-System (*ExpL-*WfMS)* ist ein Oberbegriff für eine Komposition von Instanzen bestimmter Softwarekomponenten[10]. Diese Softwarekomponenten liegen in Form von Eclipse-Plugins vor. Abbildung 9.4 visualisiert das ExpL-WfMS als Komposition von Instanzen[11] dieser Softwarekomponenten (inklusive der jeweils eingesetzten Basistechnologien). Zur besseren Lesbarkeit wird bei der Benennung der Bestandteile des ExpL-WfMS auf den Zusatz „Instanz einer Softwarekomponente“ verzichtet. So bedeutet die Bezeichnung „ExpL-*Workflow-Engine (*ExpL-*Wfe)*“ beispielsweise, dass es sich um eine Instanz der Softwarekomponente ExpL-Wfe handelt. Das ExpL-WfMS verwaltet ExpL-Wf-Definitionen und bietet hierfür die Funktionen Anzeigen, Editieren, Laden, Speichern, Instanziieren, Ausführen und Überwachen der Ausführung.

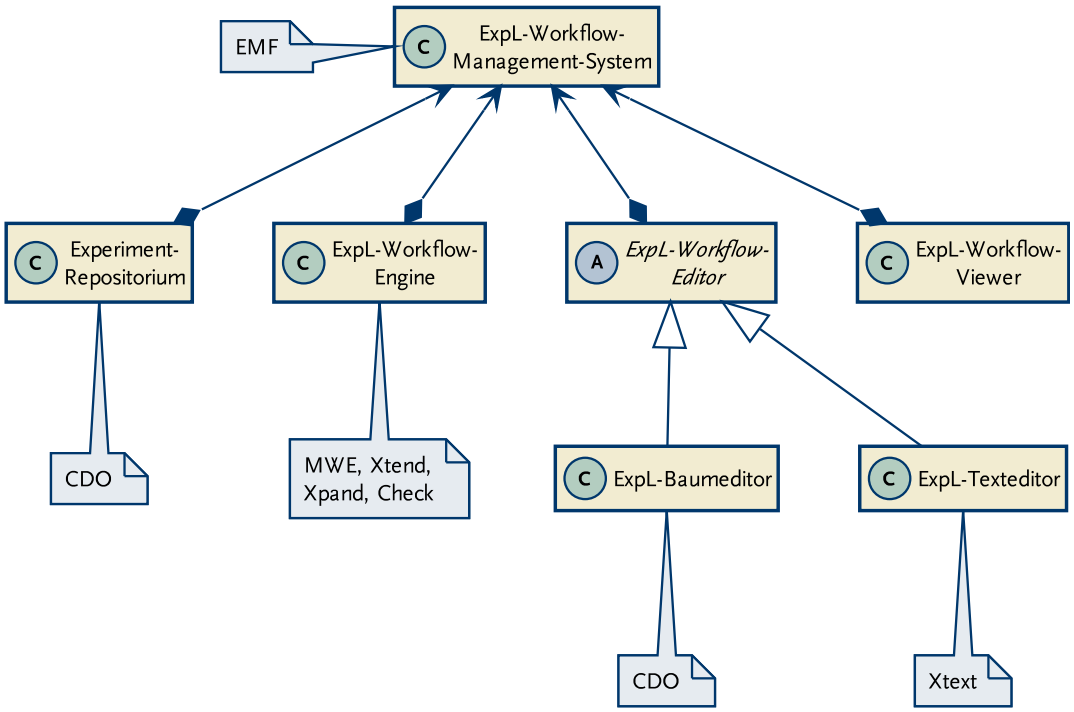


Abbildung 9.4.: Bestandteile des ExpL-Workflow-Management-System und die jeweils eingesetzten Basistechnologien (als UML-Klassendiagramm; erweitert Abbildung 9.1 auf Seite 119)

[10] Der Begriff *Softwarekomponente* wird in dieser Arbeit konform zu Szyperski folgendermaßen verwendet: ”A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.” [SGM02]. Er wird in dieser Arbeit jedoch nicht im Zusammenhang mit einem Komponentenmodell verwendet.

[11] Eine Instanz wird aus einer Softwarekomponente erzeugt und liegt im Arbeitsspeicher eines Computersystems.

9.3.1. Arbeitsablauf

Eine vollständige ExpL-Wf-Definition ist die Grundlage, um den darin beschriebenen Experimentier-Workflow mit dem ExpL-WfMS ausführen zu können. Dabei werden aus der ExpL-Wf-Definition verschiedene Artefakte automatisch transformiert, die als Eingaben für einzelne Bestandteile des ExpL-WfMS dienen.

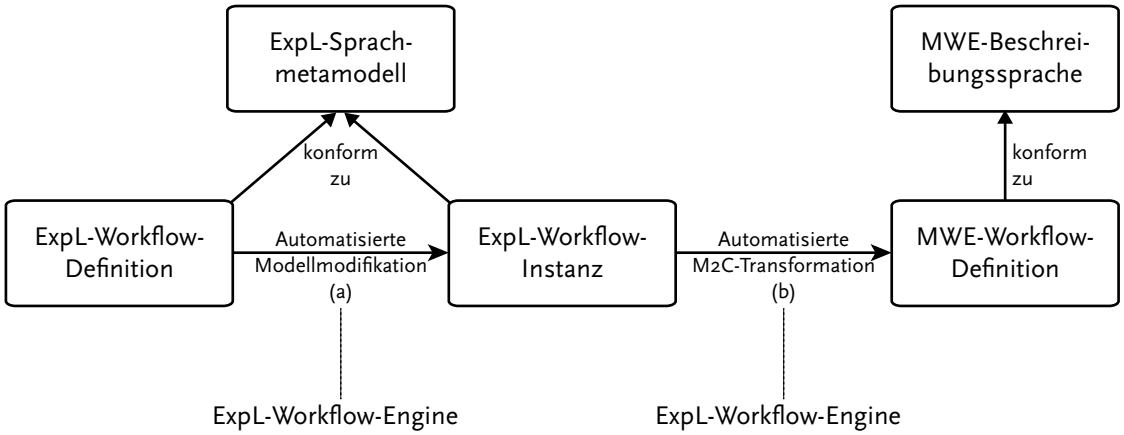


Abbildung 9.5.: Transformationen einer ExpL-Wf-Definition zur ExpL-Wf-Instanz und weiter zur MWE-Workflow-Definition

Abbildung 9.5 zeigt diese Transformationen für eine einzelne Ausführung. Aus einer ExpL-Wf-Definition erzeugt die ExpL-Wfe dabei durch Modellmodifikationen eine ExpL-*Wf-Instanz*. Sie ist ausführbar und enthält Informationen über die konkreten Experimente, die aus den Experimentplänen in der Instanziierungsphase berechnet wurden (*Instanzinformationen*). Eine ExpL-Wf-Instanz kann mehrfach ausgeführt werden, wobei sie für jede Ausführung Informationen zu Laufzeit und Monitoring (z. B. aufgetretene Fehler bei der Ausführung von Aktivitätsinstanzen) enthält (*Instanzlaufzeitinfomationen*). Eine ExpL-Wf-Instanz unterscheidet sich von der zugrundeliegenden ExpL-Wf-Definition nur durch zusätzliche, neue Elemente zur Speicherung der Instanz- und Instanzlaufzeitinfomationen. Sowohl die ExpL-Wf-Definition, als auch die ExpL-Wf-Instanz sind aus technischer Sicht Teile des selben Ecore-Modells, das konform zum ExpL-Sprachmetamodell ist.

Die Basisfunktionen einer Workflow-Engine, wie beispielsweise das Ausführen von Aktivitäten, werden durch MWE erbracht. Betrachtet man MWE als Plattform zur Ausführung von Workflows im Sinne der MDSD, so kann man die ExpL-Wf-Instanz als PIM sehen, denn sie enthält keine spezifischen Informationen über MWE.[12] Entsprechend wird von der ExpL-Wfe aus der ExpL-Wf-Instanz ein für MWE spezifisches Modell generiert (PSM), das entsprechend als *MWE-Workflow-Definition* bezeichnet wird. Die MWE-Workflow-Definition wird an MWE übergeben, dort instanziiert und ausgeführt.

[12] Die MWE-spezifische Transformationsbeschreibung wurde mittels Xpand vorgenommen und ist Bestandteil der ExpL-Wfe. Die ExpL-Wfe enthält zudem die für MWE notwendigen Erweiterungen (siehe Ab-

Abbildung 9.6 auf Seite 138 zeigt das Zusammenspiel der Bestandteile des ExpL-Workflow-Management-System beim vereinfachten Ausführen eines ExpL-Wf als Sequenzdiagramm. Der ExpL-Wf-Editor fungiert als Schnittstelle zum Experimentator.[13] Die als CDO-Client und als CDO-Server gekennzeichneten Bestandteile des ExpL-WfMS lassen sich auf physisch verschiedenen Computern betreiben. Lediglich die ExpL-Wfe und der Simulator müssen auf dem selben Computer ausgeführt werden.

In Abbildung 9.6 ist ein einfacher ExpL-Wf mit drei Aktivitäten dargestellt. Die erste Aktivität führt den Simulator (Sequenzen 9 und 10) in einer Schleife solange aus, bis jedes der n Experimente durchgeführt ist. Dabei wird jedes Experiment nur einmal abgearbeitet (ein Experimentlauf), wofür jeweils eine neue Instanz des Simulators erzeugt wird. In der zweiten Aktivität (Sequenz 11) werden die Experimentergebnisse eingelesen und gespeichert (vereinfacht in der selben Schleife dargestellt). Die Darstellung der dritten Aktivität (Sequenz 13) deutet an, dass weitere, externe Programme aufgerufen werden können, um die Experimentergebnisse zu verarbeiten.

9.3.2. Architektur

Abbildung 9.7 auf Seite 139 zeigt eine Übersicht der Architektur des ExpL-Workflow-Systems mit den verwendeten Basistechnologien. Sie ist von oben nach unten strukturiert und folgt den Transformationen der ExpL-Wf-Definition (aus Abbildung 9.5). Dabei zeigt sich von oben nach unten die konsequente Anwendung von MDSD, bei der die Spezifikation des Experimentier-Workflows unabhängig von spezifischer Technologie erfolgt und erst durch Transformationen auf spezifische Technologien abgebildet wird. So enthält die ExpL-Wf-Definition beispielsweise nur Informationen über Art und Reihenfolge der in ihr enthaltenen Aktivitäten, nicht jedoch darüber, wie diese Aktivitäten auf einer Maschine auszuführen sind. Erst durch die Transformationen hin zur MWE-Workflow-Definition und deren interpretativer Ausführung durch MWE erfolgt die Abarbeitung der ExpL-Wf-Definition.

Die verwendeten Basistechnologien des Eclipse-Technologieraumes sind grundsätzlich austauschbar, wobei der Aufwand, eine bestimmte Technologie im ExpL-Wf-System zu ersetzen, stark variiert. So sind die Konzepte im ExpL-Sprachmetamodell zwar mit Hilfe des Metametamodells Ecore formal gefasst worden, doch wären hierzu auch andere Technologien in der Lage gewesen (z. B. UML-Klassendiagramme). Weil jedoch die meisten anderen Basistechnologien auf Ecore basieren, wäre ein solcher Wechsel mit erheblichem Aufwand verbunden. Für andere Basistechnologien, wie beispielsweise Xtext, wurden die Alternativen (aus dem Eclipse-Technologieraum) bereits diskutiert.

Im Gegensatz dazu wäre ein Wechsel der Workflow-Engine leichter möglich, wobei die benötigten Aktivitäten in der neuen Workflow-Engine entweder vorhanden sein oder hinzugefügt werden müssten. Dies kann wiederum einen erheblichen Aufwand bedeuten. Ist die-

schnitt 9.1.5).

[13] Die Phasen *Planung*, *Ausführung* und *Auswertung* eines Experimentier-Workflows lassen sich auch als Vorgehensmodell des Experimentators erkennen.

se Voraussetzung erfüllt, so muss im ExpL-WfMS lediglich die Transformation (b) in Abbildung 9.7 angepasst werden. Auf diese Weise ließe sich eine Workflow-Engine verwenden, die beispielsweise auf einem Grid-Computing-Cluster arbeitet (zu einem solchen Szenario siehe Abschnitt 13).

9.3.3. ExpL-Workflow-Editoren

Das ExpL-WfMS bietet zwei Editoren, mit denen ExpL-Wf-Definitionen erstellt werden können. Der ExpL-Baumeditor hat im aktuellen Implementierungsstand einen erweiterten Funktionsumfang im Vergleich zum ExpL-Texteditor, um beispielsweise die Ausführung eines ExpL-Workflow zu starten. Zudem funktioniert der ExpL-Viewer derzeit nur in Kombination mit dem ExpL-Baumeditor.

9.3.3.1. ExpL-Baumeditor

Der ExpL-Baumeditor ist ein CDO-Client und wird durch CDO bereitgestellt. Er nutzt die Tatsache, dass jedes Element in einem Ecore-Modell eindeutig genau einem Eltern-Element zuzuordnen ist. Dies wird als *containment hierarchy* bezeichnet und lässt sich geradlinig in eine baumartige Darstellung übersetzten, bei der die einzelnen Hierarchieebenen ein- und ausgeblendet werden können.

Abbildung 9.8 auf Seite 140 zeigt ein Bildschirmfoto des ExpL-Baumeditors, der eine geladene ExpL-Wf-Definition anzeigt. Das Wurzelelement des Baumes ist `ExperimentationWorkflow`. Die Benennung eines Astes im ExpL-Baumeditor entspricht dabei dem des entsprechenden ExpL-Sprachmetamodellelementes, gefolgt vom Wert seines „name“-Attributes (oder des ersten Elementes, das vom Typ String ist). Die Attribute des aktuell im Baumeditor ausgewählten Modellelemente sind im unteren Teil der Abbildung 9.8 in der Eclipse-View „Properties“ dargestellt.

9.3.3.2. ExpL-Texteditor

Der ExpL-Texteditor wurde für die Notationssprache von ExpL mittels Xtext generiert. Hierfür wurde ein Xtext-Assistent verwendet, der aus einem vorhandenen Ecore-Modell (in diesem Fall dem ExpL-Sprachmetamodell) eine passende Grammatik für Xtext generiert. Diese generierte Grammatik enthielt viele redundante Schlüsselworte, so dass auf dieser Basis nur sehr unleserliche Texte entstehen. Deshalb wurde sie massiv verändert, um lesbarere Texte mit einer eingängigeren Syntax zu ermöglichen. Auf diese Art entstand die Grammatik, welche die Notationssprache von ExpL beschreibt. Der ExpL-Texteditor akzeptiert somit Texte, die konform zur Notationssprache von ExpL sind. Er bietet Funktionen wie Syntax-Hervorhebung, Code-Vervollständigung und Fehlerüberprüfung. Abbildung 9.9 auf Seite 140 zeigt ein Bildschirmfoto des ExpL-Texteditors, inklusive eines Outline.

Vorhandene, im Experiment-Repositorium mittels CDO verwaltete ExpL-Workflows lassen sich unter Verwendung der ExpL-Wfe als textuelle Repräsentation in der Notationssprache

von ExpL speichern. Xtext steuert dabei die Transformation des ExpL-Wf, der ein zum ExpL-Sprachmetamodell konformes Modell ist, hin zur textuellen Repräsentation, die konform zur Grammatik der Notationssprache von ExpL ist. Dabei handelt es sich um eine M2C-Transformation. Die Rückrichtung ermöglicht Xtext durch die Bereitstellung eines Parsers, der aus entsprechenden Texten ExpL-Workflows erzeugt.

Die aktuelle Implementierung des ExpL-Texteditors hat ein Problem mit der Auflösung von Referenzen innerhalb des ExpL-Wf. Deshalb müssen Referenzen als vollqualifizierte Namen (*fully qualified names*) angegeben werden, die jeweils den kompletten Pfad vom Wurzelelement `ExperimentationWorkflow` beinhalten. Soll beispielsweise auf das in Abbildung 9.9 markierte `CDORepositoryElement` mit der Bezeichnung NetTopo verwiesen werden, so muss dies in der Form expl.NetTopo erfolgen. Dies stellt aber kein konzeptionelles Problem dar und könnte durch eine eigene, selbst implementierte Referenzauflösung anstelle der von Xtext standardmäßig angebotenen gelöst werden. In dem konkreten Beispiel müssten alle Elemente innerhalb des ExpL-Wf vom Typ `CDORepositoryElement` gesammelt werden, um den Namensraum zu bilden, in dem der Bezeichner NetTopo eindeutig wäre. Aus diesem Grunde und zugunsten einer besseren Lesbarkeit wurde darauf verzichtet, in den Listings dieser Arbeit vollqualifizierte Namen bei Referenzen darzustellen.

9.3.4. ExpL-Workflow-Viewer

Abbildung 9.10 auf Seite 141 zeigt ein Bildschirmfoto des ExpL-Workflow-Viewers, der in Eclipse als eigenständige „View“ umgesetzt wurde. Der ExpL-Viewer arbeitet aktuell nur in Kombination mit dem ExpL-Baumeditor, weil eine Visualisierung nur dann erstellt wird, wenn im Baumeditor das Wurzelelement `ExperimentationWorkflow` markiert wurde. Das ist jedoch nur eine Beispielmöglichkeit, wie dem ExpL-Workflow-Viewer ein ExpL-Workflow übergeben werden kann. Eine andere Möglichkeit wäre es, direkt auf vorhandene, im Experiment-Repositorium mittels CDO verwaltete ExpL-Workflows zuzugreifen.

Der ExpL-Workflow-Viewer verfügt intern über eine M2C-Transformation auf der Basis von Xpand, die ein Skript für das Werkzeug PlantUML [20113b] erzeugt. Dieses Skript wird an PlantUML übergeben und ausgeführt.[14] Die dadurch erzeugte Visualisierung kann per Rechtsklick als Grafikdatei gespeichert werden.

Abbildung 9.10 zeigt eine Visualisierung des NetTopo-Beispiels (siehe Anhang B.3 auf Seite 178), wobei je nach Typ der Aktivitätsinstanz verschiedene Informationen hinzugefügt wurden, wie beispielsweise die Dauer der Ausführung oder der Name einer Eingabedatei. Um weitere Informationen anzuzeigen oder den Stil der Visualisierung anzupassen, wäre lediglich eine Änderung der Transformationsbeschreibung notwendig.

[14] Zur Ausführung von PlantUML wurde das PlantUML-Eclipse-Plugin für den speziellen Kontext angepasst.

9.3.5. ExpL-Workflow-Engine

Die ExpL-Wfe lädt, instanziiert und führt eine ExpL-Wf-Definition aus. Das beinhaltet den in Abschnitt 9.3.1 beschriebenen Arbeitsablauf, bei dem eine ExpL-Wf-Definition zur ExpL-Wf-Instanz und weiter zur MWE-Workflow-Definition transformiert wird. Dieser Arbeitsablauf wird innerhalb der ExpL-Wfe durch einen internen, festen MWE-Workflow definiert und ausgeführt. Dieser interne MWE-Workflow besteht aus folgenden Aktivitäten (abgebildet in Listing D.1 auf Seite 197):

1. Das ExpL-Sprachmetamodell wird geladen.
2. Eine Sitzung und eine Transaktion zu CDO werden geöffnet (CDO verwaltet das Konfigurations-Repositorium mit den ExpL-Wf-Definitionen).
3. Die auszuführende ExpL-Wf-Definition wird aus CDO geladen.
4. Eine semantische Überprüfung der geladenen ExpL-Workflow-Definition mittels Modell-Checking wird durchgeführt. Bei Fehlern wird die Ausführung der ExpL-Wf-Definition beendet.
5. Falls in der ExpL-Wf-Definition Beschränkungen für die Belegung von MUX-Eingabeparametern spezifiziert sind, werden geeignete Werte berechnet (siehe Abschnitt 8.3.6). Die Berechnung erfolgt in drei Schritten:
   a) M2C-Transformation, die ein Eingabeskript für den Constraint-Solver JaCoP erzeugt,
   b) die Ausführung von JaCoP und
   c) Einlesen der Ergebnisse von JaCoP und Speichern im ExpL-Wf.
6. Die Experimente, die durch Experimentpläne in der geladenen ExpL-Wf-Definition definiert sind, werden berechnet. Das ist ein Teil der Modellmodifikation von der ExpL-Wf-Definition zur ExpL-Wf-Instanz. Diese Modellmodifikation ist innerhalb der ExpL-Wfe in Java implementiert.
7. Commit der Transaktion. Dadurch werden die bisher durchgeführten Modifikationen an der ExpL-Wf-Definition persistent, die somit erweitert wurde und ab diesem Zeitpunkt als ExpL-Wf-Instanz bezeichnet wird.
8. Die erzeugte ExpL-Wf-Instanz wird mittels Modell-Checking überprüft. Hierbei wird vor allem geprüft, ob JaCoP eine Lösung berechnen konnte. Falls nicht, sind möglicherweise keine Experimente definiert worden, und es wird eine Warnung ausgegeben.
9. Eine MWE-Workflow-Definition wird erzeugt, welche die Aktivitäten der bisher erstellten ExpL-Wf-Instanz beinhaltet. Dies geschieht mittels einer in Xpand formulierten M2C-Transformation.
10. Die erzeugte MWE-Workflow-Definition wird an MWE übergeben und ausgeführt.

Betriebsmodi

Die ExpL-Wfe ist in Java implementiert und als eigenständiges Java-Programm außerhalb von Eclipse lauffähig (wobei es von Bibliotheken aus Eclipse und aus den Basistechnologien abhängt). Die ExpL-Wfe kann in folgenden Modi gestartet werden:

`MAP_ONLY` Die ExpL-Wfe führt die Modellmodifikationen von einer ExpL-Wf-Definition zur ExpL-Wf-Instanz durch und beendet sich. Dabei werden alle oben genannten Aktivitäten, mit Ausnahme von 9 und 10, durchgeführt. Somit kann der Experimentator im ExpL-Editor oder im ExpL-Workflow-Viewer u. a. erkennen, welche Experimente durch die Experimentpläne erzeugt wurden.

`EVALUATE_ONLY` Die ExpL-Wfe geht davon aus, dass bereits eine ExpL-Wf-Instanz ausgeführt wurde. Alle temporären Dateiartefakte, die in der ExpL-Wf-Instanz referenziert sind und die Experimentergebnisse enthalten, werden erneut eingelesen. Zudem werden alle Auswertungspläne erneut ausgeführt. Danach beendet sich die ExpL-Wfe.

`EXECUTE_ALL` Die ExpL-Wfe führt alle oben genannten Aktivitäten eins bis zehn durch und beendet sich.

`LISTEN_AND_EXECUTE` Die ExpL-Wfe startet und registriert sich bei CDO als Listener. Anschließend wartet sie, dass eine ExpL-Wf-Definition manuell vom Experimentator zur Ausführung freigegeben wurde. Sofern die dort für die Ausführung angegebene Maschine mit der übereinstimmt, auf der die ExpL-Wfe aktuell ausgeführt wird, arbeitet sie den ExpL-Wf zum darin spezifizierten Zeitpunkt ab. Anschließend wartet sie erneut auf eine freigegebene ExpL-Wf-Definition. Dies ist der einzige Modus, in dem keine Referenz auf ein bestimmte ExpL-Wf-Definition übergeben werden muss.

`DELETE_GENERATED_WF` Die ExpL-Wfe löscht alle Elemente einer ExpL-Wf-Instanz, die nicht zur ExpL-Wf-Definition gehören und beendet sich. Die verbleibende ExpL-Wf-Definition kann wiederum als Basis neuer Experimentier-Workflows dienen.

`SAVE_TEXT_MODEL` Die ExpL-Wfe transformiert eine ExpL-Wf-Instanz in die entsprechende textuelle Repräsentation (konform zur Notationssprache von ExpL).

`IMPORT_TEXT_MODEL` Die ExpL-Wfe liest eine textuelle Repräsentation (konform zur Notationssprache von ExpL) und erzeugt daraus einen ExpL-Wf, der im Konfigurations-Repositorium gespeichert wird. Dieser Modus ist aktuell nicht vollständig implementiert.

Für den komfortablen Aufruf der ExpL-Wfe wurde die Erweiterung *ExpL-Actions* für den ExpL-Baumeditor entwickelt, die ein Kontextmenü hinzufügt. Dieses Kontextmenü wird per Rechtsklick auf das Wurzelelement `ExperimentationWorkflow` aufgerufen und ist in Abbildung 9.11 dargestellt.

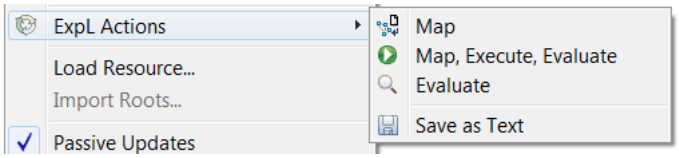


Abbildung 9.11.: Bildschirmfoto von Aufrufmöglichkeiten der ExpL-Wfe als Kontextmenü des ExpL-Baumeditors

Die Einträge Map und Evaluate des Kontextmenüs von ExpL-Actions führen zu Aufrufen der ExpL-Wfe in den Betriebsmodi `MAP_ONLY` und `EVALUATE_ONLY`. Der Menüeintrag Map, Execute, Evaluate entspricht dem Betriebsmodus `EXECUTE_ALL` und Save as Text bewirkt Aktionen des Betriebsmodus `SAVE_TEXT_MODEL`. Dabei wird jeweils eine neue Instanz einer Java-VM gestartet, in der die ExpL-Wfe ausgeführt wird.

Interprozesskommunikation

Bestimmte ExpL-Wf-Aktivitäten erfordern die Interaktion mit externen Programmen. Zu diesem Zweck startet die ExpL-Wfe solche Programme als externe Betriebssystemprozesse. Um mit diesen Betriebssystemprozessen zu kommunizieren, verwendet die ExpL-Wfe dem Aufruf mitgegebene Kommandozeilenparameterwerte, Umgebungsvariablen des Betriebssystems oder erzeugt Konfigurationsdateien. Eine Interprozesskommunikation mit laufenden Prozessen findet nicht statt.

Der verwendete Ansatz zur Interprozesskommunikation ist somit als rudimentär zu bezeichnen. In Workflows, bei denen in vielen Aktivitätsinstanzen das gleiche Programm erneut gestartet werden muss, wäre eine einmalige Instanziierung dieses Programms dann von Vorteil, wenn allein dessen Startvorgang viele Ressourcen benötigt (z. B. durch Aufbau von Datenbankverbindungen). In jeder Aktivitätsinstanz müsste somit nur mit einem bereits laufenden Betriebssystemprozess dieses Programms interagiert werden. Im Vergleich zu jeweils neuen Instanziierungen könnten dadurch Ressourcen eingespart werden (z. B. Zeit- und CPU-Bedarf).

In der aktuellen Implementierung der ExpL-Wfe startet sie externe Betriebssystemprozesse stets auf dem Computersystem, auf dem sie selbst läuft. Über geeignete Remote-Kommandos[15] des Betriebssystems wäre eine entfernte Ausführung solcher Prozesse möglich. Allerdings müsste bereits zur Entwurfszeit des ExpL-Wf bekannt sein, auf welchen Maschinen diese externen Programme ausgeführt werden sollen. Sehr viel flexibler und leistungsfähiger wäre ein Ansatz, bei dem zur Laufzeit es ExpL-Wf dynamisch herausgefunden wird, wo ein bestimmtes Programm ausgeführt werden kann.

[15] Mit der *Secure Shell (SSH)* lassen sich beispielsweise Kommandos auf einem entfernten Computersystem ausführen. SSH liefert dabei den Rückgabewert des Kommandos und Meldungen des Standardausgabekanals sowie des Standardfehlerkanals zurück.

Ein solcher Ansatz könnte durch Verwendung einer anwendungsorientierten *Middleware* umgesetzt werden, um von physischen Maschinen bzw. Rechnerknoten zu abstrahieren.[16] Die Grundidee einer Middleware in diesem Szenario ist es, vereinfacht dargestellt, dass ein bestimmtes, externes Programm eine gewisse Dienstleistung für eine Aktivität erbringt. Dieses Programm wird zusammen mit der Dienstbeschreibung (und Verwaltungsinformationen) als Dienstanbieter bei der Middleware registriert. Die ExpL-Wfe würde zur Laufzeit bei der Ausführung der entsprechenden Aktivität eine Anfrage an die Middleware stellen, wo ein solcher Dienstanbieter verfügbar ist und mit diesem, wiederum über die Middleware, interagieren, bis die erforderliche Leistung erbracht wurde. Das Konzept einer Middleware ließe sich zudem erweitern, um auch einzelne Aktivitäten dienstebasiert über die Middleware auszuführen. Damit könnte eine Workflow-Engine kreiert werden, die bei der Ausführung von Workflows sehr gut skalierte. Insbesondere wäre das der Fall, wenn ausreichend Dienstanbieter zur Verfügung stehen, etwa innerhalb eines Grid-Computing-Clusters.

9.3.6. Experiment-Repositorium

Der Begriff *Experiment-Repositorium* ist die Bezeichnung für eine Instanz der Softwarekomponente Experiment-Repositorium. Die Softwarekomponente Experiment-Repositorium ist komponiert aus drei anderen Softwarekomponenten, die im Folgenden vorgestellt werden (visualisiert in Abbildung 9.12). Ein Experiment-Repositorium besteht dabei aus einer festgelegten Anzahl von Instanzen dieser drei Softwarekomponenten.

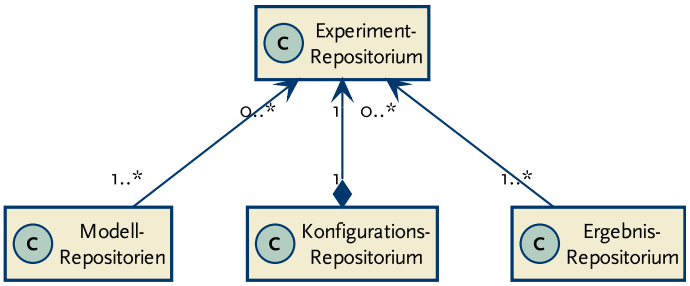


Abbildung 9.12.: Bestandteile der Softwarekomponente Experiment-Repositorium (als UML-Klassendiagramm)

9.3.6.1. Modell-Repositorium

Die Softwarekomponente Modell-Repositorium verwaltet Artefakte der zu untersuchenden Modelle (Models-Under-Experimentation). Ein MUX-Artefakt muss eindeutig und persistent aus einem Modell-Repositorium lesbar sein (z. B. mittels Persistenter-Identifier). Ein Experiment-Repositorium verfügt über mindestens eine Instanz der Softwarekomponente

[16] Beispiele für eine Middleware sind die *Common Object Request Broker Architecture (CORBA)* der OMG und das OSGi-Framework, das auf Java basiert und dem Eclipse-Technologieraum zuzuordnen ist. Aus diesem Grunde würde das OSGi-Framework zur Architektur des ExpL-WfMS passen.

Modell-Repositorium. Zum Betreiben eines Modell-Repositoriums wird existierende Technologie in Form von Quellcodeverwaltungssystemen verwendet. Das ExpL-WfMS unterstützt Git [TH05] und Subversion [sub11].

9.3.6.2. Konfigurations-Repositorium

Ein Experiment-Repositorium verfügt über genau eine Instanz der Softwarekomponente Konfigurations-Repositorium. Das Konfigurations-Repositorium im ExpL-WfMS ist repräsentiert durch eine Instanz eines CDO-Servers, der mit dem ExpL-Sprachmetamodell konfiguriert wurde, so dass er dazu konforme Modelle verwaltet (die ExpL-Wf-Definitionen und -Instanzen).

9.3.6.3. Ergebnis-Repositorium

Typischerweise werden Experimentergebnisse als Bestandteil der ExpL-Wf-Instanz und somit im Konfigurations-Repositorium gespeichert. Je nach Art und Umfang der Experimentergebnisse kann es allerdings sinnvoller sein, andere Speichersysteme zu verwenden, wie beispielsweise ein RDBMS oder eines der unterstützten Quellcodeverwaltungssysteme.

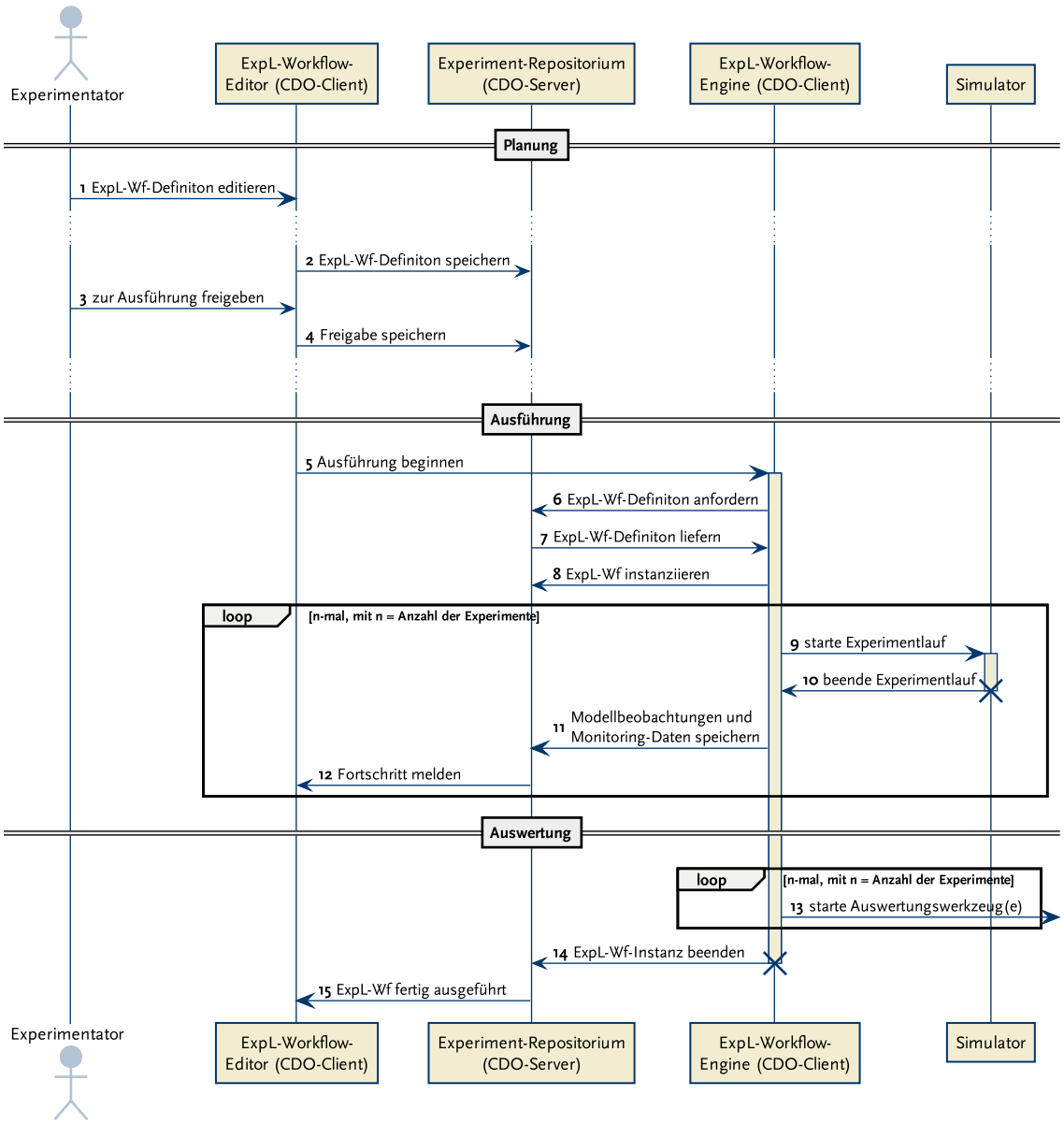


Abbildung 9.6.: Zusammenspiel der Bestandteile des ExpL-Workflow-Management-System beim vereinfachten Ausführen eines ExpL-Workflow (als UML-Sequenzdiagramm)

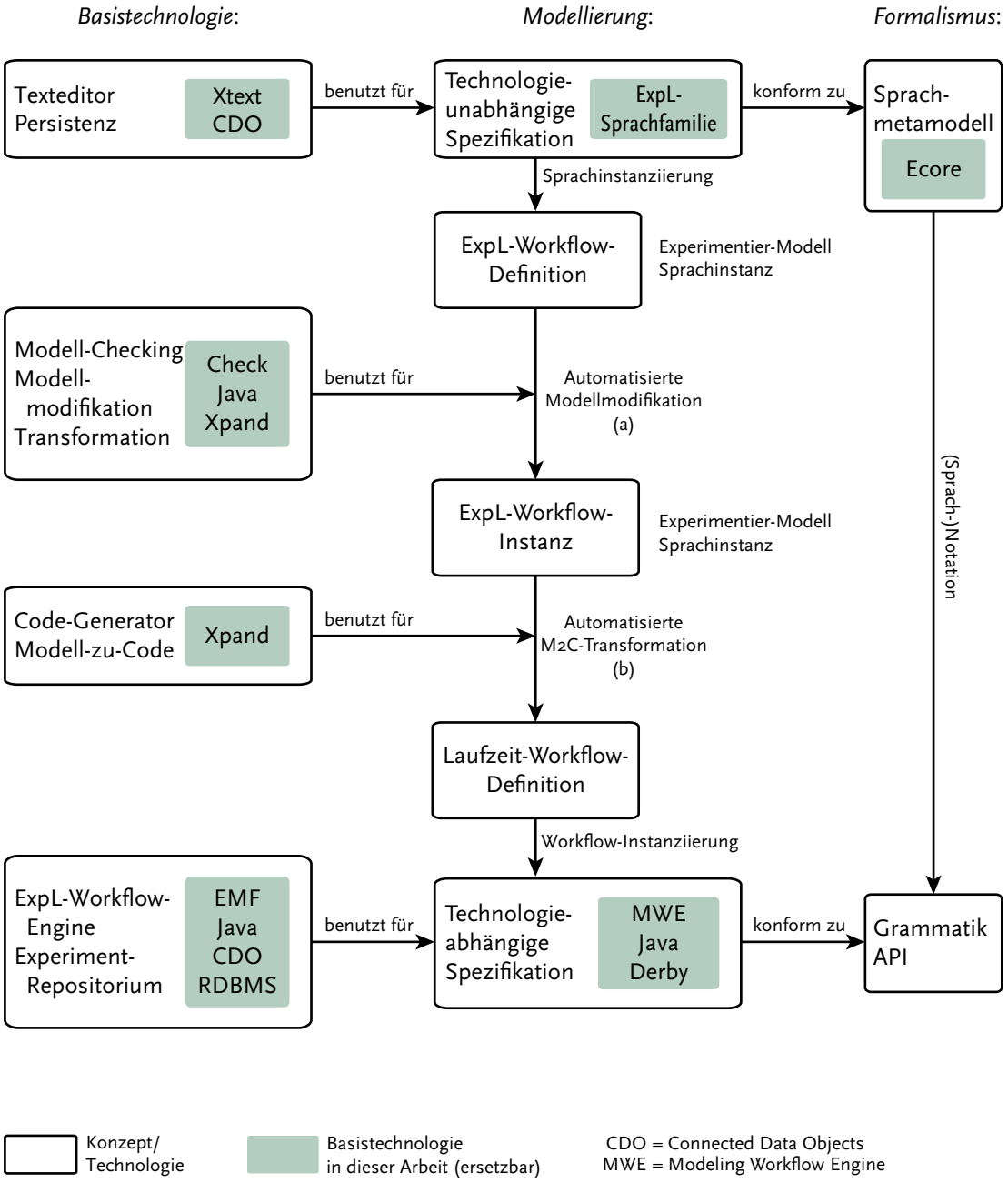


Abbildung 9.7.: Übersicht der Architektur des ExpL-Workflow-Systems mit den verwendeten Basistechnologien

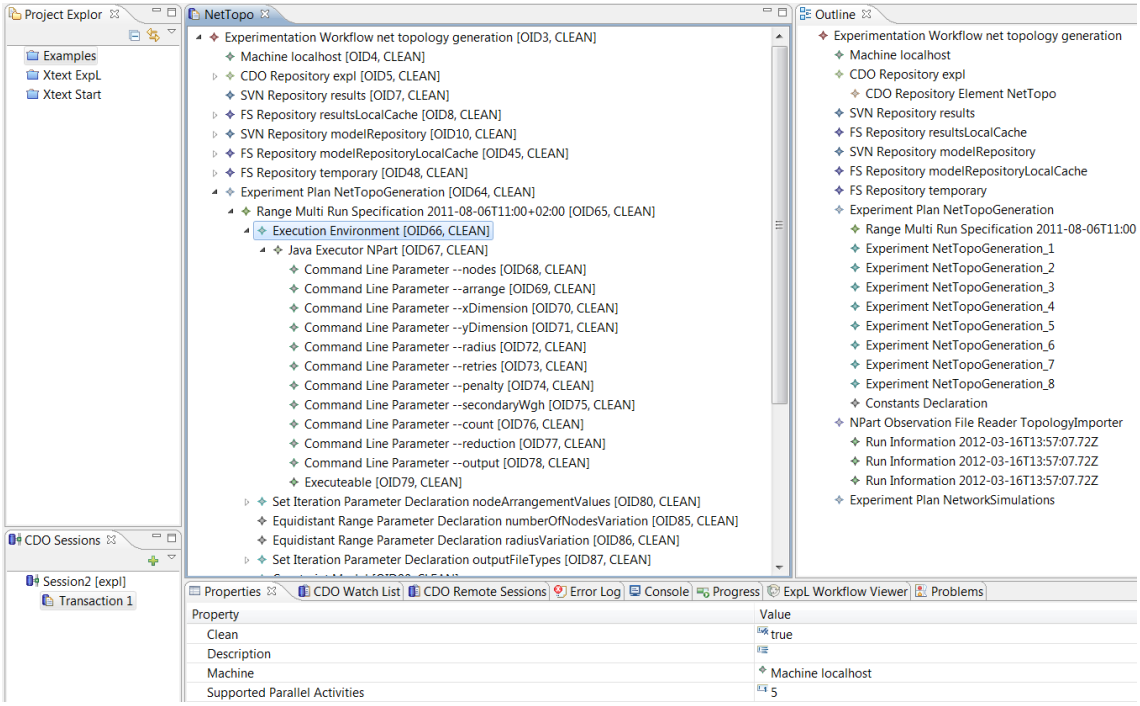


Abbildung 9.8.: Bildschirmfoto des ExpL-Baumeditors, der das NetTopo-Beispiel zeigt (siehe Anhang B.3 auf Seite 178)

Abbildung 9.9.: Bildschirmfoto des ExpL-Texteditors, der das NetTopo-Beispiel zeigt (siehe Anhang B.3 auf Seite 178)

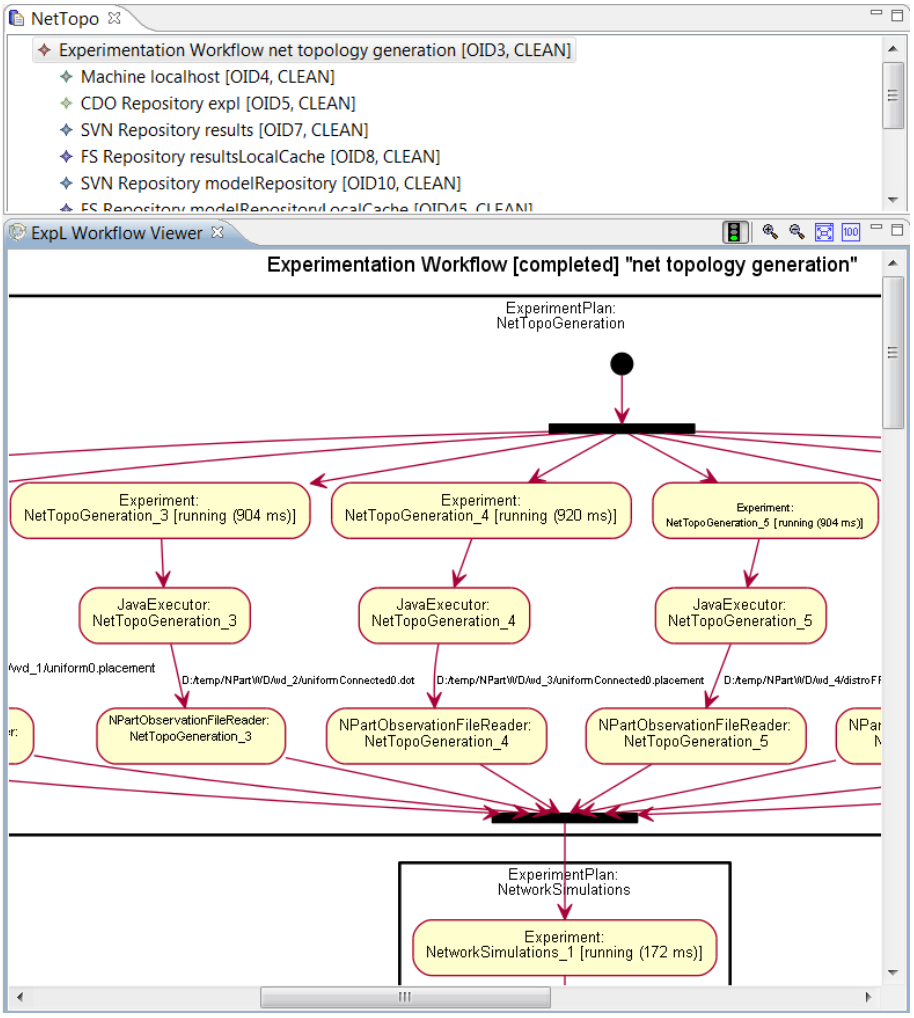


Abbildung 9.10.: Bildschirmfoto des ExpL-Workflow-Viewers (in Eclipse als „View“), der das NetTopo-Beispiel zeigt (siehe Anhang B.3 auf Seite 178)

10. Anwendung von ExpL in Fallstudien

In diesem Kapitel 10 sind Experimentier-Workflows der in Kapitel 7 auf Seite 71 beschriebenen Fallstudien dargestellt. Durch die Analyse des jeweiligen, traditionellen Experimentier-Prozesses ergaben sich Verbesserungen und Erweiterungen, so dass für jede Fallstudie zunächst der verbesserte Experimentier-Prozess dargestellt wird. Darauf aufbauend wird exemplarisch gezeigt, wie der verbesserte Experimentier-Prozess mit ExpL umgesetzt wurde (die vollständigen ExpL-Modelle sind im Anhang C ab Seite 183 zu finden). Ein Erfahrungsbericht schließt den jeweiligen Abschnitt der Fallstudie ab.

10.1. Experimentier-Workflows in der SLEUTH-Fallstudie

10.1.1. Ansatz

Die unübersichtliche Vermengung von Aspekten in den Phasen Modellierung/Planung, Ausführung und Auswertung durch die SLEUTH-Implementierung zeigte sich als ungeeignete Basis für den in der Fallstudie angestrebten Vergleich zwischen SLEUTH* und SLEUTH*P (Abschnitt 7.1.3). Deshalb wurde die SLEUTH-Implementierung nach verschiedenen, konzeptionellen Aspekten aufgeteilt (visualisiert in Abbildung 10.1 auf der folgenden Seite). Hierbei wurde der in Abschnitt 8.1 auf Seite 101 vorgestellte sprachbasierte Ansatz angewendet. Im Einzelnen wurden folgende separate Modelle erstellt:

SLEUTH*(P) Der Zelluläre Automat zur Beschreibung des urbanen Landnutzungswandels wurde von Theisselmann als SLEUTH*(P) konform zu der von ihm entwickelten, domänenspezifischen Sprache ECAL re-implementiert [TKK+10, TDF09].

Experimentier-Workflow Die Kalibrierungsphase von SLEUTH*(P) wurde mittels ExpL modelliert. Zu diesem Zweck wurde ECAL mit ExpL gekoppelt[1] [KTF09].

Analyse Um eine Analyse der georäumlichen Modellbeobachtungen durchführen zu können, wurde von Theisselmann eine einfache Sprache für georäumliche Auswerteverfahren (GIS-DSL) erstellt. Diese GIS-DSL wurde mit ECAL und ExpL gekoppelt, so dass sie in den Experimentier-Workflow eingebunden werden konnte [KTF09].

[1] In dieser Sprachkopplung wurde ExpL um die notwendigen Konzepte aus ECAL erweitert.

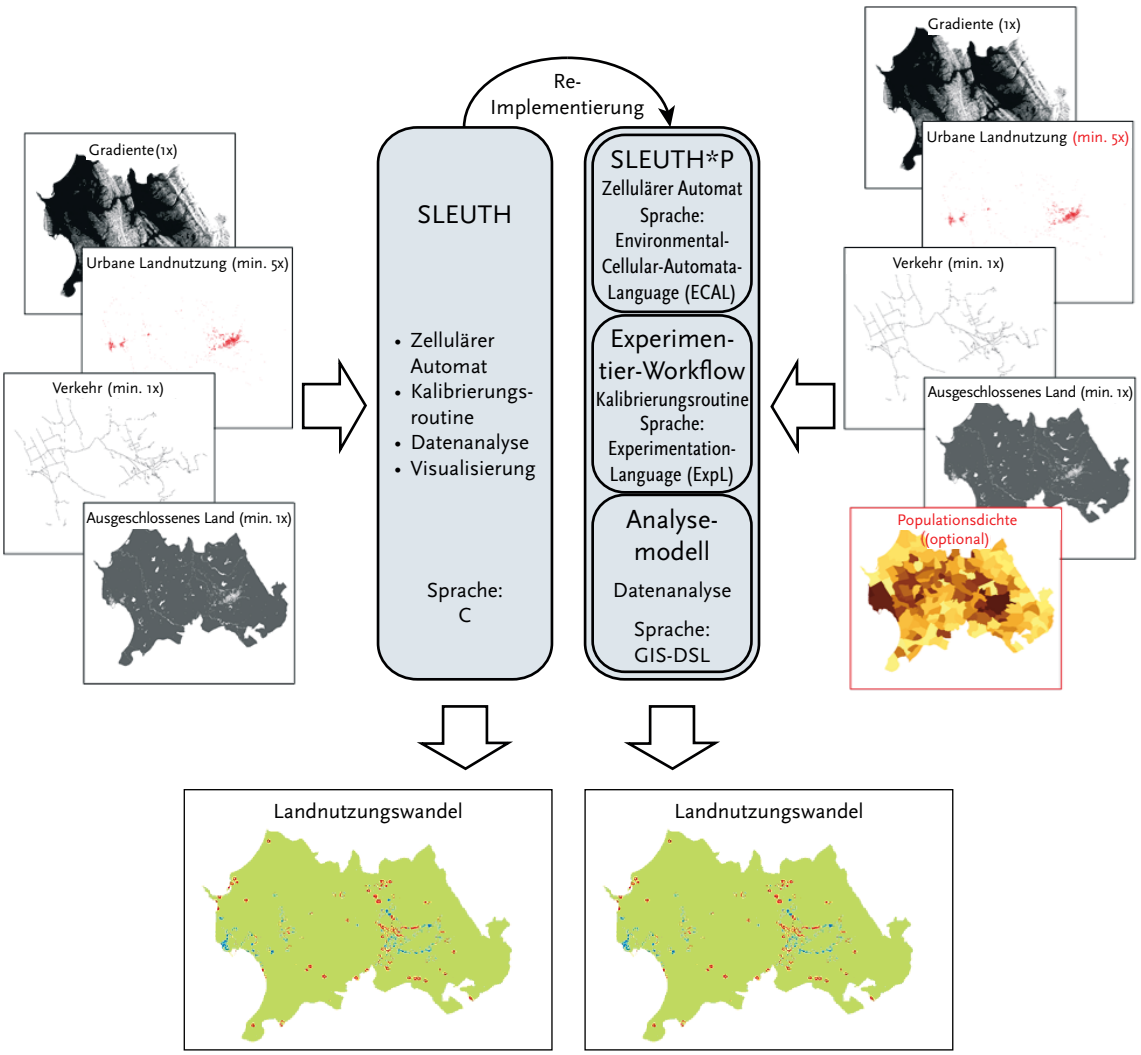


Abbildung 10.1.: Von SLEUTH zu SLEUTH*P mit Aufteilung nach Aspekten, umgesetzt durch gekoppelte DSLs

10.1.2. Experimentier-Prozess und Beispiel

Abbildung 10.2 visualisiert die Kalibrierungsphase der Modellklasse SLEUTH*(P) als Experimentier-Prozess und kennzeichnet darin solche Aktivitäten mit dem Stereotyp «ExpL», die mittels ExpL beschrieben wurden. Im Vergleich zum traditionellen Experimentier-Prozess (Abbildung 7.2 auf Seite 75) wurde ExpL eingesetzt, um Aktivitäten für die Variation der Eingabeparameterbelegung und für die Ausführung der SLEUTH*(P)-Modelle zu beschreiben. Die Auswertung der Resultate mittels der GIS-DSL ist eine neu hinzugekommene Aktivität, die neben der traditionellen Auswertung der Statistikmaßwerte stattfindet. Die Referenzdatensätze wurden vorgegeben und unverändert verwendet. Ihre Erzeugung sollte als Bestandteil des Experimentier-Workflows aufgenommen werden, um dessen Transparenz zu erhöhen.

Die ExpL-Workflow-Definition umfasst drei Bereiche (die typisch für einen ExpL-Wf sind,

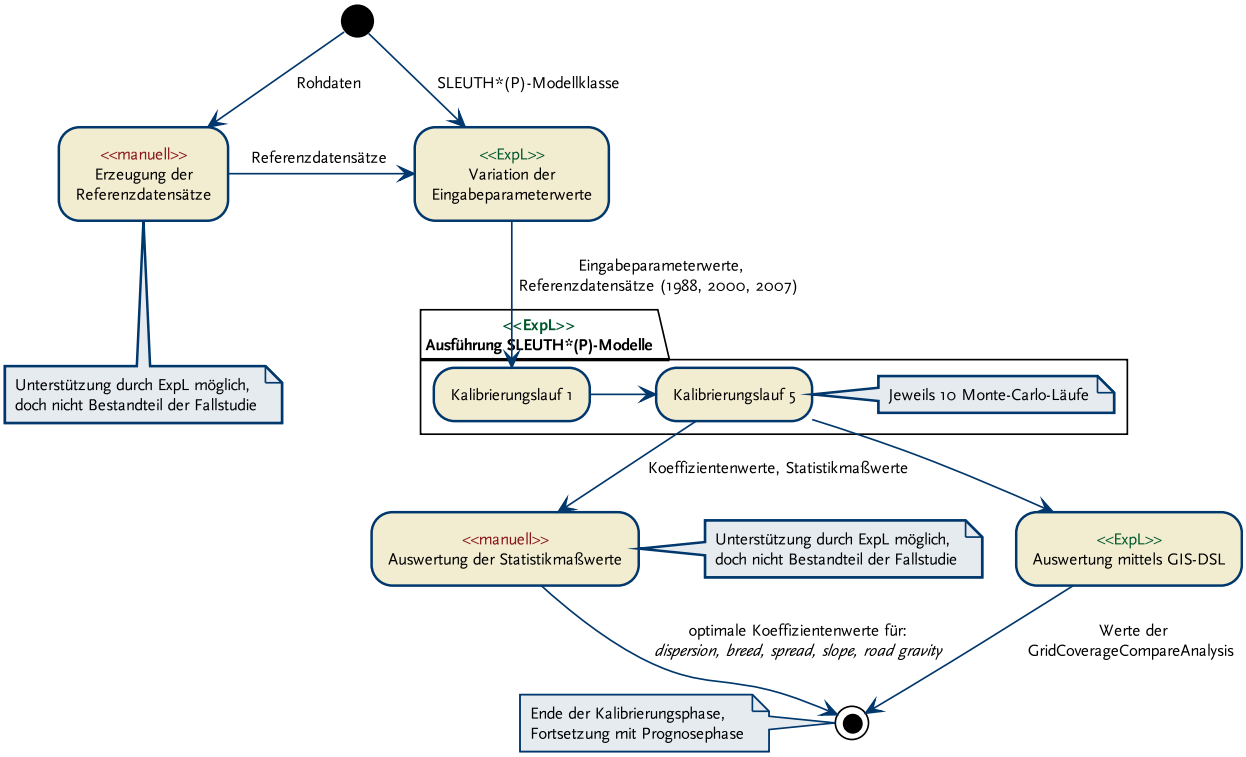


Abbildung 10.2.: Experimentier-Prozess von SLEUTH*(P): Kalibrierungsphase unter Verwendung von ExpL (als UML-Aktivitätsdiagramm; traditioneller Prozess in Abbildung 7.2 auf Seite 75)

siehe Abschnitt 8.3.7):

1. Die Modelleingabeparameter und Modellausgangsgrößen der SLEUTH*(P)-Modellklasse werden definiert (siehe MUX-Deklaration im Anhang in Listing C.2 auf Seite 184).
2. Die Variation der Eingabeparameterbelegung des MUX wird durch einen `ExperimentPlan` definiert (Listing 10.1 auf der nächsten Seite).
3. Die GIS-DSL-basierte Auswertung der Experimentergebnisse wird durch zwei Elemente des Typs `EvaluationPlan` definiert (Listing C.4 auf Seite 187 im Anhang).

SLEUTH*(P) wurde mit Referenzdatensätzen aus den Jahren 1988, 2000 und 2007 für die Region Tirana (Albanien) kalibriert. In [Can02] wurde gezeigt, dass bereits fünf Kalibrierungsläufe genügen, um eine gute Annäherung an die optimale Kombination der Koeffizientenwerte zu erreichen. In jedem Kalibrierungslauf wurde der Lee-Sallee-Index berechnet.[2] Der höchste Indexwert kennzeichnet dabei die diesbezüglich optimalen Eingabeparameterwerte.

Jeder Kalibrierungslauf besteht aus zwei Phasen (*coarse* und *fine*):

[2] Das SLEUTH*(P)-Modell führt intern 10 Monte-Carlo-Läufe durch, wobei der Seed-Eingabeparameter (siehe Listing C.2 auf Seite 184, Zeile 18) den Startvektor des Zufallszahlengenerators initialisiert.

1. Durch die Verwendung von größeren, äquidistanten Schrittweiten wird der gesamte Parameterraum abgedeckt bei gleichzeitig reduziertem Berechnungsaufwand im Vergleich zu kleineren Schrittweiten. Bei fünf Parametern mit Werten in einem Bereich von jeweils 0 bis 100 und einer äquidistanten Schrittweite von 25 ergeben sich $5^5 = 3125$ mögliche Kombinationen an Eingabeparameterbelegungen.
2. Die 10 besten (bezüglich des Lee-Sallee-Indexes) Kombinationen werden verwendet, um mit kleineren Schrittweiten eine verfeinerte Betrachtung vorzunehmen.

Listing 10.1 zeigt den `ExperimentPlan` zur Beschreibung der Parameterbelegungsvariation in der ersten Phase. Er ist Teil der Experimentier-Workflow-Definition in ExpL, die in Listing C.1 auf Seite 183 dargestellt ist.

```
ExperimentPlan SLEUTH*(P)-Kalibrierung for SLEUTH_Cal_Coarse_constCoeff0.ecal {
 RepeatedRangeMultiRun startAt "2010-09-05T11:00:00+02:00" parallel plannedExecutions
     5 {
  executionEnvironment { /* dargestellt in Listing C.3 auf Seite 186 */ }

  domainVariables { /* referenziert MUX-Deklaration, dargestellt in Listing C.2 auf
      Seite 184 ab Zeile 6 */
   EquidistantRange varyDispersionCoeff { vary DispersionCoeff "0.0" -> "100.0" #
       "25.0" },
   EquidistantRange varyBreedCoeff { vary BreedCoeff "0.0" -> "100.0" # "25.0" },
   EquidistantRange varySpreadCoeff { vary SpreadCoeff "0.0" -> "100.0" # "25.0" },
   EquidistantRange varySlopeCoeff { vary SlopeCoeff "0.0" -> "100.0" # "25.0" },
   EquidistantRange varyRoadCoeff { vary RoadCoeff "0.0" -> "100.0" # "25.0" }
  }

  constants { /* referenziert MUX-Deklaration, dargestellt in Listing C.2 auf
      Seite 184 ab Zeile 6 */
   FileConstantInputParameter constSlope { Slope = SLEUTH/albania/slope.tif },
   FileConstantInputParameter constExcluded { Excluded =
       SLEUTH/albania/exclusion7.tif },
   FileConstantInputParameter constUrban { Urban = SLEUTH/albania/urb88_2.tif },
   FileConstantInputParameter constTransportation { Transportation =
       SLEUTH/albania/roads88_3.tif },
   ConstantInputParameter constCRITICAL_HIGH { CRITICAL_HIGH = "100.0" },
   ConstantInputParameter constCRITICAL_LOW { CRITICAL_LOW = "0.0" },
   ConstantInputParameter constBOOM { BOOM = "1.01" },
   ConstantInputParameter constBUST { BUST = "0.9" },
   ConstantInputParameter constSlopeSensitivity { SlopeSensitivity = "0.1" },
   ConstantInputParameter constRoadSensitivity { RoadSensitivity = "0.01" },
   ConstantInputParameter constMIN_SLOPE_RESISTANCE { MIN_SLOPE_RESISTANCE = "0.01" },
   ConstantInputParameter constMIN_ROAD_GRAVITY { MIN_ROAD_GRAVITY = "0.01" },
   ConstantInputParameter constMIN_DIFFUSION { MIN_DIFFUSION = "0.01" },
   ConstantInputParameter constMIN_SPREAD { MIN_SPREAD = "0.01" },
   ConstantInputParameter constMIN_BREED { MIN_BREED = "0.01" },
   ConstantInputParameter constMAX_SLOPE_RESISTANCE { MAX_SLOPE_RESISTANCE = "100" },
   ConstantInputParameter constMAX_ROAD_GRAVITY { MAX_ROAD_GRAVITY = "100" },
   ConstantInputParameter constMAX_DIFFUSION { MAX_DIFFUSION = "100" },
   ConstantInputParameter constMAX_SPREAD { MAX_SPREAD = "100" },
   ConstantInputParameter constMAX_BREED { MAX_BREED = "100" },
```

```
34      ConstantInputParameter constCRITICAL_SLOPE { CRITICAL_SLOPE = "15" },
        ConstantInputParameter constMAX_ROAD_VALUE { MAX_ROAD_VALUE = "10" },
36      FileConstantInputParameter const_t0 { t0 = SLEUTH/albania/roads88_3.tif },
        FileConstantInputParameter const_t1 { t1 = SLEUTH/albania/roads00_3.tif },
38      ConstantInputParameter constSeed { Seed = "1" }
    }
40  }
}
42

Legende: Schlüsselwort, Identifier, Referenz, "String", /* Kommentar */
```

Listing 10.1: Textuelle Repräsentation des ExpL-Sprachelementes `Experiment Plan`, angewendet zur Beschreibung der Variation der Koeffizientenbelegung im Workflow der SLEUTH-Fallstudie (Ausschnitt aus Listing C.1 auf Seite 183)

10.1.3. Erfahrungen

Die Erfahrungen aus der SLEUTH-Fallstudie belegen, dass der gezeigte, sprachbasierte Ansatz zur Beschreibung des SLEUTH-Experimentier-Prozesses anwendbar und vorteilhaft ist [LTK+12, KTF09]. Die Aufteilung in verschiedene Modelle, die jeweils mit einer formalen Beschreibung in Form einer DSL erfolgt, fördert die Verständlichkeit und Nachnutzbarkeit. Die Vorteile einer DSL für die Modellierung demonstrierte Theisselmann bei der Erweiterung von SLEUTH* zu SLEUTH*P [TKK+10].

Die Vorteile eines sprachbasierten Ansatzes zeigen sich in der SLEUTH-Fallstudie insbesondere bei der Nachnutzbarkeit und Verständlichkeit (gehören zu den Anforderungen an Experimentier-Prozesse, siehe Abschnitt 7.4). Die Nachnutzbarkeit zeigte sich bei der Wiederverwendung des Exp-Wf von SLEUTH* auch bei SLEUTH*P, was durch die Flexibilität der ExpL-Sprachbeschreibung erreicht wird. Die Verständlichkeit verbesserte sich auf der Ebene des Exp-Wf durch die Verwendung von Workflow-Konzepten, die den Ablauf der Experimente klar strukturieren und die entstandenen Artefakte (z. B. Modellergebnisse) zuorden, wann und durch welche Aktivität sie entstanden sind.

10.2. Experimentier-Workflows in der Nano-Fallstudie

Die Nano-Fallstudio konnte aufgrund von personellen Veränderungen in der beteiligten Physik-Arbeitsgruppe um Prof. Benson nicht beendet werden.[3] Als Ergebnis der Zusammenarbeit entstand die domänenspezifische Sprache *Nano-DSL*[4] zur Beschreibung von in der Physik-Arbeitsgruppe gängigen, optischen Nanostrukturen und darauf basierenden, einzelnen

[3] Nach dem Ausscheiden von Janik Wolters nahm die Möglichkeit zur Zusammenarbeit in der Physik-Arbeitsgruppe stetig ab, bis zuletzt keine Kontaktaufnahme mehr gelang und die Fallstudie abgebrochen werden musste.

[4] Die Nano-DSL wurde in der gemeinsamen Arbeitsgruppe entwickelt und von Martin Schmidt konzeptionell ausgearbeitet und implementiert im Rahmen seiner Masterarbeit [Sch11] und METRIK-SHK-Tätigkeit.

Experimenten [Sch11, WSKF11]. Experimentserien (und Experimentier-Workflows) sind damit nicht beschreibbar (siehe Abschnitt 7.2.3). Um eine solche Beschreibung zu ermöglichen, war eine Kopplung von Nano-DSL und ExpL geplant, die jedoch nur partiell umgesetzt werden konnte.

10.2.1. Experimentier-Prozess und Beispiel

Abbildung 10.3 auf Seite 155 zeigt den angestrebten, verbesserten Experimentier-Prozess, wobei im Vergleich zum traditionellen Experimentier-Prozess (Abbildung 7.4 auf Seite 97) veränderte oder neu hinzugefügte Aktivitäten mit dem Stereotyp «neu» gekennzeichnet sind. Die Modellierung der Experimente geschieht hierbei nicht mehr im proprietären Format von FDTD Solutions, sondern mittels der domänenspezifischen Sprachen Nano-DSL und ExpL (erste Aktivität, linke Seite in Abbildung 10.3). In Verbindung mit einer Transformationsvorschrift dieser Sprachkombination hin zu einem werkzeug-spezifischen Format wird erreicht, dass das Simulationswerkzeug austauschbar ist (zweite Aktivität, linke Seite in Abbildung 10.3). Dadurch wird beispielsweise eine automatisierte Ausführung in FDTD Solutions und/oder Meep unterstützt, ohne das in der Nano-DSL vorliegende MUX zu verändern.[5]

Eine Beschreibung im Format der Nano-DSL besteht aus folgenden fünf Blöcken, die hier überblicksartig vorgestellt werden sollen (eine detaillierte Darstellung liefert [Sch11] und Listing C.5 auf Seite 190 im Anhang zeigt ein vollständiges Beispielmodell):

`SETUP` Deklaration von Attributen als modellweit referenzierbare, numerische Werte in der Form `val ValName = Value`.

`UNIT` Festlegung von modellweit gültigen Einheiten für Länge, Zeit, Frequenz und Dämpfung. Es kann jeweils aus einer Menge von vordefinierten Einheiten ausgewählt werden.

`STRUCTURE` Definiert die optische Nanostruktur des photonischen Kristalls. `STRUCTURE` besteht aus der Beschreibung von Material, Membran (`Slab`), Gitter (`Lattice`) und verschiedener, geometrischer Objekte (z. B. Quader, Kegel, Kugel und Pyramide).

`SIMULATION` Definiert Attribute, wie die Modellzeitdauer der Simulation, die Position und Größe des Simulationsraumes (auf der optischen Nanostruktur).

`SOURCES` Beschreibung eines Dipols (magnetisch oder elektrisch) zur Erzeugung eines elektromagnetischen Impulses. `SOURCES` besitzt Attribute wie beispielsweise Position, Öffnungs- Schließungswinkel und Breitband- und/oder Schmalbandfilter.

`MONITORS` Spezifiziert Position und Art eines Monitors aus vordefinierten Typen (Zeit-, Spektral- oder Feldmonitor). Ein Monitor definiert, welche Modellbeobachtungen aufgezeichnet werden sollen.

Durch diese Blockstruktur wird eine Trennung der Beschreibungen von Struktur (`SETUP`, `UNIT`, `STRUCTURE`) und Experiment (`SIMULATION`, `SOURCES`, `MONITORS`) ermöglicht. Es

[5] In der aktuellen Implementierung wird lediglich das Simulationswerkzeug FDTD Solutions als Ausführungsumgebung unterstützt.

existiert jedoch keine Beschreibungsmöglichkeit für Experimentserien, wie sie durch die Variation von Attributbelegungen der Nano-DSL-Experimentbeschreibung entstehen können. Jede mögliche Variation zieht eine Änderung des Nano-DSL-Modells nach sich. Um beispielsweise einen Dipol an drei verschiedenen stellen zu positionieren, sind drei Experimente mit drei Nano-DSL-Modellen notwendig. Dabei besteht kein explizit beschriebener Zusammenhang zwischen diesen Modellen und deren Ergebnissen, obwohl sie inhaltlich zur selben Experimentserie gehören.

Tabelle 10.1 auf der folgenden Seite zeigt, wie Sprachelement der Nano-DSL mit Sprachelemente von ExpL zusammenhängen. Die strukturbeschreibenden Elemente (`SETUP, UNIT, STRUCTURE`) der bisherigen Nano-DSL sollen im Folgenden als *Nano*-DSL* verstanden werden und bilden ein *MUX**. Die experimentbeschreibenden Elemente (`SIMULATION, SOURCES, MONITORS`) der bisherigen Nano-DSL werden durch Elemente der Sprache ExpL ausgedrückt. Dadurch wird eine Trennung der Konzepte erreicht, wie sie in Kapitel 8 auf Seite 101 beschrieben ist.

Der Übergang von der Nano-DSL zur Kombination von Nano*-DSL und ExpL wurde vollständig konzeptionell erarbeitet, jedoch wegen des vorzeitigen Endes der Fallstudie lediglich in Ansätzen implementiert: Es existiert ein Assistent zum Import vorhandener Nano-DSL-Modelle in ExpL, der die experimentbeschreibenden Elemente in ExpL als MUX-Eingabeparameter deklariert.[6]

10.2.2. Erfahrungen

Nach Ansicht der beteiligten Domänen-Experten aus der Physik eignet sich eine textuelle Beschreibung ihrer Experimentier-Prozesse sehr gut. Ein Grund hierfür liegt darin, dass die Physiker häufig vorhandene Beschreibungen abändern, beispielsweise, indem sie eine neue Lochstruktur einfügen oder eine vorhandene verschieben. Textuell lässt sich das schneller und präziser umsetzen als mit graphischen Werkzeugen. Die Präferenz einer textuellen Beschreibung zeigt sich auch in der Nutzungsweise der bisherigen Werkzeuge zur Strukturbeschreibung, die sowohl eine graphische als auch eine textuelle Repräsentation bieten. Eine Visualisierung der Nanostruktur wird lediglich als hilfreich, aber nicht als zwingend notwendig betrachtet. Die Verwendung von DSLs und die damit einhergehende Konzentration auf die notwendigen Konzepte ermöglicht einerseits eine Loslösung von konkreten Werkzeugen und andererseits die Erschließung von sich ergänzenden Funktionen in unterschiedlichen Werkzeugen mit Hilfe lediglich einer abstrakten Beschreibung des Experimentier-Prozesses.

[6] Der Importer (Paket `expl.eclipse.xtext.ui.wizard.ecore2xtext` in den Quellen von ExpL, Abschnitt D.2 auf Seite 202) wurde von Martin Schmidt im Rahmen seiner METRIK-SHK-Tätigkeit implementiert.

Nano-DSL	ExpL
SETUP	Attribute modellierbar als Standardwert für eine Eingabeparameterdeklaration des MUX oder als Konstanten (ConstantInputParameter als Teil des ExperimentPlan).
UNIT	Jede Eingabeparameterdeklaration des MUX kann mit einer Einheitendeklaration versehen werden.
STRUCTURE	Referenziert als MUX-Artefakt (mit spezieller Sprache nano).
SIMULATION	Attribute werden deklariert als Eingabeparameter des MUX. Ihre Werte sind modellierbar unter domainVariables (Teil des ExperimentPlan), wenn eine Variation gewünscht wird oder andernfalls als Konstanten (ConstantInputParameter als Teil des ExperimentPlan).
SOURCES	Attribute werden deklariert als Eingabeparameter des MUX. Ihre Werte sind modellierbar unter domainVariables (Teil des ExperimentPlan), wenn eine Variation gewünscht wird oder andernfalls als Konstanten (ConstantInputParameter als Teil des ExperimentPlan).
MONITORS	Attribute werden deklariert als Eingabeparameter des MUX. Ihre Werte sind modellierbar unter domainVariables (Teil des ExperimentPlan), wenn eine Variation gewünscht wird oder andernfalls als Konstanten (ConstantInputParameter als Teil des ExperimentPlan).

Tabelle 10.1.: Relationen zwischen Elementen der Sprachen Nano-DSL und ExpL

10.3. Experimentier-Workflows in der SOSEWIN-Fallstudie

Im Besonderen gegenüber den anderen Fallstudien dieser Arbeit werden in der SOSEWIN-Fallstudie verschiedene Ausführungsumgebungen aktiv verwendet. Um auf diese Besonderheit einzugehen, wird im Folgenden Abschnitt 10.3.1 beispielhaft ein textuell notierter ExpL-Wf für eine Experimentserie in simulativer Ausführungsumgebung gegeben, und anschließend wird gegenübergestellt, welche Änderungen an der Workflow-Definition notwendig sind, um die gleiche Experimentserie in der Realwelt auf dem Prototypnetzwerk auszuführen.

10.3.1. Experimentier-Prozess und Beispiel

Das Beispiel in Listing 10.2 zeigt die textuelle Repräsentation einer ExpL-Workflow-Definition zur Beschreibung einer SOSEWIN-Simulationsserie mittels ODEMx (bezeichnet als Set-Up 2a auf Seite 83 und Abbildung 7.7 auf Seite 99). Die zugrundeliegende Experimentserie Hypozentrum ist beschrieben auf Seite 86.

Das in Abschnitt 7.3.1 auf Seite 82 beschriebene Modell des SOSEWIN-Alarmierungsprotokolls ist das MUX in der hier dargestellten Experimentserie (in diesem Fall das plattformunabhängige Modell des SOSEWIN-Alarmierungsprotokolls). Das MUX wird durch ein Artefakt repräsentiert, welches in Zeile 38 von Listing 10.2 eindeutig und persistent referenziert ist.

Die Eingabeparameter des MUX sind definiert in Zeile 40: Hypozentrum, Netzwerktopologie, Wellendaten und Magnitude.

In der ersten Aktivität (Zeile 53) des dargestellten ExpL-Wf wird aus dem plattformunabhängigen MUX ein für die ODEMx-Simulationsplattform abhängiges MUX-Programm erstellt, wobei ein `makefile` die notwendigen Aktionen enthält. Das daraus resultierende Artefakt `SingleNode.exe` wird im `ExperimentPlan` zur Ausführung deklariert (Zeile 68). Es wird zudem beschrieben, wie die Eingabeparameter mit konkreten Werten belegt werden sollen (Zeile 62), indem Kommandozeilenoptionen des Artefakts `SingleNode.exe` verwendet werden.

Der `ExperimentPlan` beschreibt die Variationen der Eingabeparameterbelegungen (ab Zeile 72) und die Werte der Konstanten (Zeile 80). Die Position des Hypozentrums soll auf einer Strecke zwischen zwei georäumlichen Punkten in einem Abstand von 5 km variiert werden (Zeile 73). Dies ergibt bei einer Distanz von 40 km zwischen den gegebenen Punkten 8 Experimente. Die Deklaration zur Variation der Netzwerktopologie (Zeile 76) referenziert einen hier nicht dargestellten `ExperimentPlan` (ein Beispiel hierfür ist Listing B.4 auf Seite 179 im Anhang, das 4 Experimente definiert).[7]

In der Experimentserie Hypozentrum sind Magnitude (Zeile 81) und die verwendeten seismischen Wellendaten (Zeile 82) konstant. Die Experimentserie besteht somit aus $8 * 4 = 32$ Experimenten. In der letzten Aktivität (Zeile 86) des dargestellten ExpL-Wf werden die in den Experimenten erzeugten Log-Dateien in das Experiment-Repositorium gespeichert. Es handelt sich hierbei um einen speziellen Typ von Aktivität, der auf das Ausgabeformat des MUX-Programms zugeschnitten ist.

```
1  ExperimentationWorkflow {
   description "SOSEWIN Beispiel"
3  resource = SOSEWIN
   resultRepository = SynthetischeWellendaten /* referenziert Deklaration in Zeile 11 */
5  resources {
     Machine "experimentserver.informatik.hu-berlin.de" { /* ... */ }
7  } // Ende resources
   repositories {
9    CDORepository experimentRepositorium {
       ConfigurationRepository URI = "tcp://experimentserver.informatik.hu-berlin.de:2036"
11     elements { CDORepositoryElement SOSEWIN { }  }
     },
13   SVNRepository SynthetischeWellendaten {
       ConstantDataRepository URI = "svn+ssh://pandora.informatik.hu-berlin.de
           /vol/marvin-vol4/svn/ES-SyntheticData"
15     workingDirectory SOSEWINModelRepositoryLocalCache
       elements { SVNRepositoryElement Wellendaten { } }
17   },
     FSRepository SOSEWINModelRepositoryLocalCache {
```

[7] Die Ausführung des `ExperimentPlan` aus Listing B.4 auf Seite 179 erzeugt in 4 Experimenten 4 Topologien mit 50 und 100 Knoten, die jeweils einen Funkradius von 50 m und 100 m besitzen (die Formate dot und ns2 nicht betrachtet). Auf die für das SOSEWIN-Alarmierungsprotokoll notwendige Gruppenstruktur innerhalb der Netzwerktopologie soll an dieser Stelle zur Vereinfachung nicht eingegangen werden.

```
    ModelRepository URI = "file:///export/sosewin/simulators"
    elements {
      FSRepositoryElement "SingleNode.exe",
      FSRepositoryElement makefile
    }
  },
  FSRepository SynthetischeWellendatenLocalCache {
    TemporaryRepository URI = "file:///temp"
    elements { FSRepositoryElement Wellendaten }
  },
  RDBMSRepository EMS_DB {
    ResultRepository URI = "jdbc:derby://jupiter.informatik.hu-berlin.de"
    user sosewin_devel password = EncryptedPassword { value "jnnU/8B1I1Cn571Wg3Tovg==" }
    elements { RDBMSRepositoryElement ems { } }
  },
  SVNRepository SOSEWINModelRepository {
    ModelRepository URI = "svn+ssh://pandora.informatik.hu-berlin.de/projekte/svn/repos/ES/AlarmingProtocol"
    workingDirectory SOSEWINModelRepositoryLocalCache
    elements {
      SVNModelRepositoryElement "SingleNode/SingleNode.rdp" {
        modelName "SOSEWIN Simulator"
        inputParameters {
          InputParameterDeclaration Hypozentrum as Point3D { },
          InputParameterDeclaration Netzwerktopologie as ExpNetworkTopology,
          InputParameterDeclaration Wellendaten as ExpSet type ExpReal,
          InputParameterDeclaration Magnitude as ExpReal unit Mw
        }
        language ODEMx
      },
      SVNRepositoryElement "SingleNode/makefile" { }
    }
  }
} // Ende repositories
tasks {
  MakefileScriptTask "Build Simulator" { /* nicht implementiert, doch analog zu AntScriptTask */
    outputMatArtifacts ( SingleNode.exe ) /* referenziert Deklaration in Zeile 21 */
    file makefile /* referenziert Deklaration in Zeile 22 */
    target "Simulator"
  },
  ExperimentPlan "Experimentserie Hypozentrum" for SingleNode/SingleNode.rdp {  /* referenziert Deklaration in Zeile 38 */
    RepeatedRangeMultiRun startAt "2012-08-06T11:00+02:00" parallel {
      executionEnvironment {
        supportedParallelActivities 3 binary ODEMxExecutor "ODEMx Simulator" {
          parameters {
            "--hypocentre" = Variation Hypozentrum, /* referenziert Deklaration in Zeile 73 */
            "--magnitude" = Konstante Magnitude, /* referenziert Deklaration in Zeile 81 */
            "--networktopology" = Variation Netzwerktopologie, /* referenziert
```

```
                Deklaration in Zeile 76 */
            "--data" = Konstante Wellendaten /* referenziert Deklaration in Zeile 82 */
          }
          executable Executeable { file SingleNode.exe }
        }
        machine experimentserver.informatik.hu-berlin.de /* referenziert Deklaration in
            Zeile 6 */
      }
      domainVariables {
        EquidistantRange "Variation Hypozentrum" {  /* nicht implementiert für Datentyp
            Point3d */
          vary Hypozentrum "40.60 28.42 -10.1" -> "41.00 28.98 -10.1" # "5 km"
        }
        VariationByExperimentPlan "Variation Netzwerktopologie" {
          using SynthesizeTopology /* referenziert hier nicht dargestellten
              ExperimentPlan, wie z.B. in Listing B.4 auf Seite 179 */
        }
      }
      constants {
        ConstantInputParameter "Konstante Magnitude" { Magnitude = "6.5" },
        FileConstantInputParameter "Konstante Wellendaten" {  Wellendaten from file
            Wellendaten }
      }
    }
  },
  SOSEWINObservationFileReader LogfilesImporter { /* nicht implementiert, doch analog
      zu NPartObservationFileReader */
    experimentPlan Experimentserie Hypozentrum /* referenziert Deklaration in Zeile 58
        */
  }
 } // Ende tasks
}

Legende: Schlüsselwort, Identifier, Referenz, "String", /* Kommentar */
```

Listing 10.2: Textuelle Repräsentation einer SOSEWIN-Workflow-Definition in ExpL zur Beschreibung einer Simulationsserie mittels ODEMx, Set-Up 2 (Seite 83) und Experimentserie Hypozentrum (Seite 86)

Listing 10.3 auf Seite 156 zeigt die notwendigen Anpassungen der ExpL-Wf-Definition, um den Ausführungskontext von Simulation zu Realwelt zu ändern. Im Vergleich zu Listing 10.2 auf Seite 151 wird in Aktivität eins (Listing 10.3, Zeile 2) ein anderes `makefile`-Ziel aufgerufen („Realwelt“ anstelle von „Simulation“), so dass ein für die Realwelt-Ausführungsumgebung plattformabhängiges Modell erzeugt wird, welches vom `ExperimentPlan` referenziert wird.

Im `ExperimentPlan` (Zeile 7) ändert sich zudem die Referenz zu der Maschine[8], auf der die

[8] Es wird angenommen, dass die hier referenzierte Maschine ein in der SOSEWIN-Architektur als Gateway-Knoten bezeichneter Knoten ist, der per Internet erreicht werden kann. Zudem läuft auf den SOSEWIN-Knoten ein Middleware-Dienst „Software-Deployment-Service“, der für die Verteilung des erzeugten, platt-

Experimente ausgeführt werden sollen (Zeile 14). Weitere Änderungen am `ExperimentPlan` sind nicht notwendig, so dass das Konzept der Artefakttrennung hier erfolgreich angewendet wurde. Im Vergleich zu Listing 10.2 auf Seite 151 wird eine zusätzliche Aktivität benötigt: Der `LogfilesCollector` (Zeile 19) sammelt die im SOSEWIN auf jedem Knoten entstandenen Log-Dateien ein. Dabei verwendet er die im `ExperimentPlan` referenzierte Maschine als Zugangspunkt zum SOSEWIN. Die Semantik des `LogfilesImporter` (Zeile 22) verändert sich durch den zuvor auszuführenden `LogfilesCollector`, so dass dessen gesammelte Log-Dateien verarbeitet werden.

10.3.2. Erfahrungen

Die gezeigte Konzeption des Einsatzes von ExpL für die SOSEWIN-Fallstudie beruht auf fundierten Erfahrungen im Aufbau des Experimentier-Prozesses und dessen Umsetzung durch EMS. Praktische Erfahrungen in der Ausführung von SOSEWIN-Experimenten durch das ExpL-WfMS sind jedoch nicht vorhanden, da die Semantik der zu ergänzenden Aktivitäten und Aktionen, wie sie im Abschnitt 10.3.1 gezeigt wurden, nicht implementiert worden ist.

formabhängigen Modells und der synthetischen Wellendaten zuständig ist [FKE+09c, S. 10].

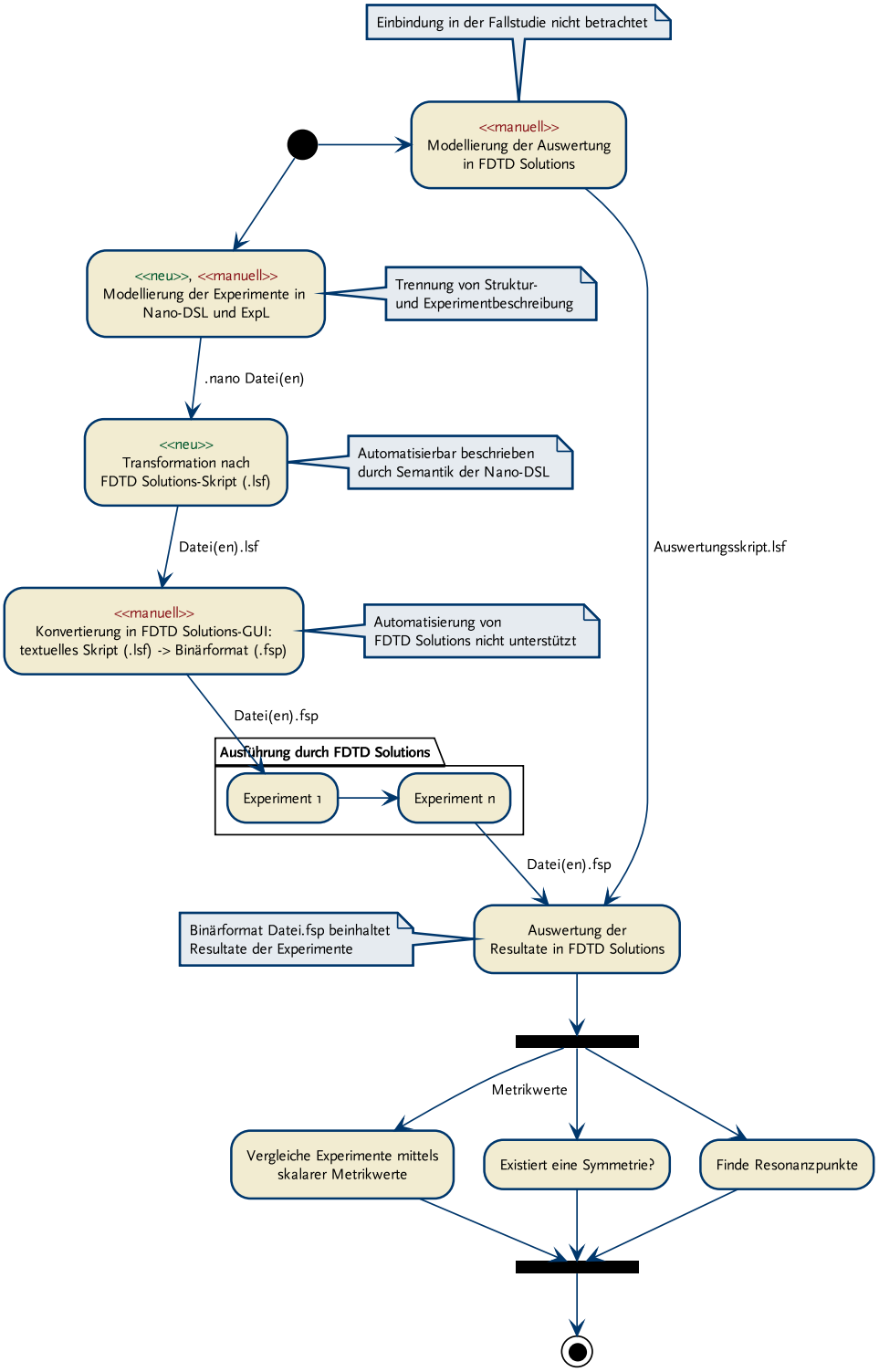


Abbildung 10.3.: Angestrebter Experimentier-Prozess mit optischen Nanostrukturmodellen (als UML-Aktivitätsdiagramm; traditioneller Experimentier-Prozess in Abbildung 7.4 auf Seite 97)

```
tasks {
  MakefileScriptTask "Build Executeable" { /* nicht implementiert, doch analog zu
      AntScriptTask */
    outputMatArtifacts ( AP ) /*+ anderes Ausgabeartefakt */
    file makefile
    target "Realwelt" /*+ anderes Makefile-Ziel */
  },
  ExperimentPlan "Experimentserie Hypozentrum" for SingleNode/SingleNode.rdp {  /*
      referenziert Deklaration in Zeile 38 */
    RepeatedRangeMultiRun startAt "2012-08-06T11:00+02:00" {
      executionEnvironment {
        binary SOSEWINExecutor "SOSEWIN Realwelt Ausführung" {
          parameters { /* unverändert im Vergleich zu Listing 10.2 auf Seite 151 */ }
          executable Executeable { file AP } /*+ durch MakefileScriptTask mit anderem
              Ziel erzeugt */
        }
        machine gateway.sosewin.koeri.boun.edu.tr  /*+ referenziert erreichbaren
            Gateway-Knoten zum Zugriff auf die SOSEWIN-Middleware */
      }
      /* weitere Definition unverändert im Vergleich zu Listing 10.2 auf Seite 151 */
    }
  },
  SOSEWINObservationFileCollector LogfilesCollector { /*+ nicht implementiert;
      hinzugefügt im Vergleich zu Listing 10.2 auf Seite 151 */
    experimentPlan Experimentserie Hypozentrum /* referenziert Deklaration in Zeile 7 */
  },
  SOSEWINObservationFileReader LogfilesImporter { /* nicht implementiert, doch analog
      zu NPartObservationFileReader */
    experimentPlan Experimentserie Hypozentrum /* referenziert Deklaration in Zeile 7 */
  }
} // Ende tasks

Legende: Schlüsselwort, Identifier, Referenz, "String", /* Kommentar */, /*+ Kommentar
    für Veränderung */
```

Listing 10.3: Veränderte Aktivitäten der ExpL-Wf-Definition für SOSEWIN zur Beschreibung einer realweltlichen Experimentserie, Set-Up 2a (Seite 83) und Experimentserie Hypozentrum (Seite 86) im Vergleich zu Listing 10.2 auf Seite 151

Teil III.

Diskussion, Zusammenfassung und Ausblick

Die Zukunft soll man nicht voraussehen wollen, sondern möglich machen.

Antoine de Saint-Exupéry

11. Diskussion

Die Zielsetzung dieser Arbeit war die Überprüfung der in Abschnitt 1.3 genannten Hypothese. Für diesen Zweck wurde das ExpL-Workflow-System mit der domänenspezifischen Sprache ExpL entwickelt. Eine zentrale Diskussionsfrage ist folglich, ob für das Experimentieren mit computerbasierten Modellen die passenden Konzepte erkannt und in ExpL umgesetzt wurden, so dass das Experimentieren mittels ExpL transparent und automatisierbar beschrieben werden kann, oder ob nicht bessere oder zusätzliche Konzepte existieren. Der Frage nach zusätzlichen Konzepten als Erweiterungen von ExpL wird in Kapitel 13 nachgegangen.

Es konnte exemplarisch gezeigt werden, dass die in ExpL realisierten Konzepte hinsichtlich der Anwendung in den drei Fallstudien vollständig sind. Dies gelang durch eine vollständige Beschreibung der Experimentier-Prozesse in den jeweiligen Fallstudien in Form von automatisierbaren und transparenten ExpL-Workflows. Die drei Fallstudien stammen nicht nur aus verschiedenen Anwendungsdomänen (Seismologie, Geographie und Physik), sondern wurden auch aus interdisziplinären Forschungsgruppen und realen Projekten heraus entwickelt. ExpL-Workflows in verschiedenen Anwendungsdomänen erfolgreich eingesetzt zu haben, demonstriert die Flexibilität des zugrundeliegenden, sprachbasierten Ansatzes, mit dem Spezifika einer bestimmten Anwendungsdomäne berücksichtigt werden können. Dies konnte zudem ebenfalls in zwei kleineren Beispielen gezeigt werden (Bossel-Fischfang aus der Literatur und NetTopo im Anhang B.3). Vom methodischen Standpunkt aus betrachtet, wäre eine möglichst hohe Anzahl von Fallstudien wünschenswert. Es ist jedoch aufgrund der bisherigen Erfahrungen zu vermuten, dass weitere Fallstudien (aus unterschiedlichen Anwendungsdomänen) die Notwendigkeit der im ExpL-Workflow-System eingebauten Flexibilität bestätigen.

ExpL entstand aufgrund einer Analyse der Methodik von Modellierung & Simulation und den durchgeführten Fallstudien. Man kann somit kritisch anmerken, dass dieselben Fallstudien sowohl für die Ermittlung von Anforderungen, als auch zur Evaluierung des daraus entwickelten Ansatzes herangezogen wurden. Bei eingehender Betrachtung ergibt sich daraus jedoch kein methodisches Problem. Die allgemeinen, domänenübergreifenden Konzepte für das Experimentieren mit Modellen (umgesetzt in ExpL-Kern) wurden durch die Analyse der Methodik von M&S gewonnen und erwiesen sich als gleichermaßen in allen Fallstudien (und Beispielen) anwendbar. Aus den Fallstudien ergab sich, dass es Spezifika des Experimentierens in den jeweiligen Domänen gibt, die diese allgemeinen Konzepte ergänzen. Diese Erkenntnis führte zur Anwendung eines sprachzentrierten Ansatzes durch die Sprachkopplung von Domain-Specific-Languages (für Konfiguration, Ausführung und Auswertung), die jeweils nur die spezifischen Konzepte der Experimentier-Domäne hinzufügen. Die leichte Erweiterbarkeit von ExpL für eine bestimmte Experimentier-Domäne ist dabei eine Stärke dieses Ansatzes. Weil die Fallstudien aus verschiedenen Domänen stammen, kann ein solcher,

sprachzentrierter Ansatz mit ihnen evaluiert werden.

In den Fallstudien zeigte sich zudem, dass sich die dort durchgeführten Experimentier-Prozesse selbst verändern, wenn Schwachstellen beseitigt werden, die durch Analysen zu Beginn der Fallstudien aufgefallen sind, oder wenn neue Möglichkeiten genutzt werden, die sich durch den Einsatz von ExpL ergeben. Deshalb waren die Experimentier-Prozesse, die zur Aufstellung von Anforderungen an das Experimentieren mit Modellen führten, unterschiedlich zu denen, die später mit ExpL automatisiert wurden. Solche Unterschiede belegen auch, dass Experimentieren durch agile, sich weiter entwickelnde Prozesse charakterisiert ist.

Technischer Ansatz

In diesem Abschnitt werden Aspekte des technischen Ansatzes und der verwendeten Technologien im ExpL-Workflow-System diskutiert.

Sprachkopplung

Die Sprachkopplung der Domain-Specific-Languages in der ExpL-Sprachfamilie findet derzeit pragmatisch auf der Basis eines gemeinsamen Metamodells statt. Technisch wäre eine Kopplung auf der Basis von mehreren Metamodellen (einem pro DSL) und Referenzen zwischen ihnen besser. Dadurch könnten die einzelnen Metamodelle separat bearbeitet werden und die Übersichtlichkeit wäre höher (vergleichbar mit der Aufteilung von Funktionen eines Programms auf mehrere Dateien).

Notationssprache von ExpL

Die Notationssprache von ExpL wird durch eine Grammatik definiert, die in der Beschreibungssprache von Xtext vorliegt. Diese Grammatik wurde initial aus dem ExpL-Sprachmetamodell abgeleitet (durch ein von Xtext bereit gestelltes Assistenzsystem) und manuell angepasst (um eine Syntax mit vielen, redundanten Schlüsselworten zu vermeiden). Änderungen des ExpL-Sprachmetamodells müssen somit manuell in der Grammatik der Notationssprache von ExpL nachgezogen werden – die Grammatik neu zu generieren würde bedeuten, die bisher gemachten, manuellen Anpassungen zu verlieren. Ein anderes Vorgehen könnte dieses Problem abschwächen: Neben dem ExpL-Sprachmetamodell wird ein zweites Sprachmetamodell zur Definition der Notationssprache von ExpL verwendet, aus dem die Grammatik generiert werden kann. Alle Veränderungen, die bisher manuell an der Grammatik durchgeführt wurden, müssten in diesem Ansatz (manuell) innerhalb einer unidirektionalen Transformation vom ExpL-Sprachmetamodell zum Sprachmetamodell der Notationssprache von ExpL untergebracht werden. Dadurch wären manuelle Anpassungen der Grammatik nicht mehr notwendig.

Multi-Modelling

Eine konzeptionell andere Art der technischen Realisierung zur Erstellung der Notationssprache von ExpL ist die Anwendung von *Multi-Modelling* [Hes09]. Bei der im vorhergehenden Absatz beschriebenen Methode, zwei Sprachmetamodelle zu verwenden, wurde eine unidirektionale Transformation zwischen diesen Sprachmetamodellen vorgeschlagen. Änderungen an dem Sprachmetamodell der Notationssprache von ExpL führen somit nicht zu Änderungen am ExpL-Sprachmetamodell (und würden bei dessen Änderung durch erneute Ausführung der unidirektionalen Transformation gelöscht werden). Beim Multi-Modelling wird eine bidirektionale (nicht-bijektive) Abbildung zwischen diesen Sprachmetamodellen definiert, wodurch Änderungen an einem Metamodell zu dem jeweils anderen übertragen werden. Eine solche Abbildung kann jedoch schwierig umzusetzen sein. Weil man die Notationssprache von ExpL jedoch als Sicht (*view*) auf die Sprache ExpL verstehen kann, ist ein spezieller Ansatz möglich, der aktuell erforscht wird: die Verwendung von sogenannten *Linsen* [Wid11]. Dabei muss ein Metamodell eine Abstraktion des anderen sein – dies trifft zu, weil die Notationssprache von ExpL nicht mehr Konzepte enthält, als die Sprache ExpL selbst.

Migration von ExpL-Workflows

Bei Veränderungen am ExpL-Sprachmetamodell müssen u. a. auch vorhandene ExpL-Workflows migriert werden, um sie weiter im ExpL-Wf-System verwenden zu können (andernfalls sind sie jedoch immer noch menschenlesbar und besitzen dokumentarischen Wert). Für einige Fälle ist dies trivial. Beispielsweise wenn ein Sprachelement dem ExpL-Sprachmetamodell hinzugefügt wird, kann es noch nicht in vorhandenen ExpL-Wf-Definitionen enthalten sein. Für andere Fälle ist eine Migration automatisch möglich (z. B. beim Umbenennen von Sprachelementen). Sie kann jedoch auch nur manuell möglich sein (z. B. wenn ein Sprachelement entfernt wurde, dessen Information mittels anderer Sprachelemente ausgedrückt werden soll). Diese Probleme bei Modellanpassungen aufgrund eines veränderten Metamodells sind generell bekannt und wurden beispielsweise von Wachsmuth in [Wac07] beschrieben. Das ExpL-Wf-System bietet hierfür keine Migrationsunterstützung an.

Performanz

Zur Performanz des ExpL-Wf-Systems wurden keine Untersuchungen gemacht. Performanz im Umgang mit sehr großen Datenmengen, die als Ergebnisse von Experimenten entstehen können, war kein Entwurfskriterium des Systems. Das ExpL-Wf-System bietet standardmäßig ein Ergebnis-Repositorium (siehe Abschnitt 9.3.6.3), das die Experimentergebnisdaten innerhalb der ExpL-Wf-Instanz speichert, die sie erzeugt hat. ExpL-Wf-Instanzen werden von CDO verwaltet, das so konstruiert ist, dass es auch nur Teile der Daten einer ExpL-Wf-Instanzen bereitstellen kann. Dadurch wird u. A. vermieden, alle Experimentdaten im Hauptspeicher halten zu müssen, obwohl nur bestimmte Daten eines einzelnen Experimentes benötigt werden.

Für das Experimentieren mit Sensornetzwerken, wie dem HWL, bei dem in kurzer Zeit sehr viele Experimentdaten gespeichert werden müssen, hat sich CDO als zu langsam erwiesen [SZFK12]. Für solche Anwendungsfälle kann im ExpL-Wf-System auch ein anderer Typ von Ergebnis-Repositorium genutzt werden (z. B. ein RDBMS). In [SZFK12] wird *EMFFrag* als alternatives System zu CDO vorgestellt – konzeptionell könnte *EMFFrag* auch mit dem ExpL-Wf-System genutzt werden.

12. Zusammenfassung

Modellierung & Simulation (M&S) bedeutet die Erstellung und experimentelle Untersuchung von computerbasierten Modellen, die Struktur und Verhalten eines Systems abstrahiert beschreiben. Oftmals sind dabei die betrachteten Systeme entweder (noch) nicht physisch existent oder durch anderweitige Gründe einer realen Untersuchung schwer zugänglich. Deshalb untersucht man stellvertretend Modelle dieser Systeme. In vielen naturwissenschaftlichen Disziplinen ist M&S daher ein unverzichtbares Instrument des Erkenntnisgewinns.

Umso wichtiger sind das Design und das Management von Modellexperimenten. In der Praxis erfordern aussagekräftige Modellexperimente das systematische Untersuchen des Modelleingabeparameterraumes, wodurch sehr viele Modellexperimente geplant, durchgeführt, überwacht und ausgewertet werden müssen. Oftmals müssen Modelleingabedaten erst aus heterogenen Datenbeständen gewonnen werden. Nach der Modellausführung werden die Modellbeobachtungsdaten ausgewertet, wobei sie aggregiert und statistisch sowie grafisch aufbereitet werden können. Dadurch ergeben sich Ketten von Datenverarbeitungsaufgaben, die jeweils spezifische Werkzeuge erfordern, die in verschiedenen Konfigurationen und Versionen genutzt werden können. Das Experimentieren mit computerbasierten Modellen ist ein sich dynamisch verändernder, iterativer Prozess mit variabel komponierbaren Teilaufgaben.

Das Ziel dieser Arbeit war es, die folgende Hypothese zu überprüfen: Das Experimentieren mit computerbasierten Modellen kann transparent und automatisierbar beschrieben werden, indem man geeignete (und erweiterte) Workflow-Konzepte mit Hilfe einer domänenspezifischen Sprache formalisiert und diese Sprache mittels eines Computersystems anwendet. Diese Hypothese konnte bestätigt werden, indem ein solches Computersystem in Form des metamodellbasierten ExpL-Workflow-Systems entwickelt und in Fallstudien und Beispielen angewendet wurde. Zum ExpL-Workflow-System gehört die domänenspezifische Sprache ExpL, mit der Experimentier-Prozesse als ExpL-Workflows formal, transparent und automatisierbar beschrieben werden können.

Der Begriff *Transparenz* fasst die Anforderungen Verständlichkeit, Nachnutzbarkeit und Reproduzierbarkeit zusammen, die an Beschreibungen von Experimentier-Prozessen gestellt werden. Diese Anforderungen wurden durch eine Analyse der Methodik von Modellierung & Simulation identifiziert und in drei durchgeführten Fallstudien bestätigt. In den drei Fallstudien aus unterschiedlichen Domänen zeigte sich, dass domänenspezifische Konzepte in der jeweiligen Domäne existieren, in der experimentiert wird (z. B. eine Variation der Knotendichte in Sensornetzwerken). Die Transparenz des Experimentier-Workflows wird deutlich erhöht, wenn diese Konzepte auch als Beschreibungsmittel in ExpL zur Verfügung stehen. Um dies zu ermöglichen, wurde ein sprachzentrierter Ansatz gewählt, bei dem die Aspekte Konfiguration, Ausführung und Auswertung von Modellexperimenten in verschiedene, gekoppelte

Domain-Specific-Languages aufgeteilt wurden. Diese Domain-Specific-Languages lassen sich an die entsprechende Domäne anpassen, in der experimentiert wird.

Die Anpassbarkeit an eine Experimentier-Domäne wird auch durch den Einsatz von Workflow-Konzepten (z. B. die Komponierbarkeit von Aktivitäten) gefördert. Die Semantik neuer ExpL-Sprachelemente kann dabei auf vorhandene Aktivitäten abgebildet werden oder durch neue Aktivitäten hinzugefügt werden. Das ExpL-Workflow-System ist zudem in einer Form technologieunabhängig, so dass ExpL-Workflows prinzipiell auch in einem Workflow-System ausgeführt werden können, das möglicherweise bereits in der Experimentier-Domäne eingesetzt wird. Dadurch ließen sich bereits existierende, spezielle Aktivitäten eines solchen Workflow-Systems nutzen, indem neue ExpL-Sprachelemente eingeführt werden, die diese Aktivitäten auf einem höheren Abstraktionsniveau beschreibbar machen.

Das ExpL-Workflow-System kann flexibel auf eine Experimentier-Domäne angepasst werden. Dies wurde durch die folgenden drei Konzepte erreicht: den sprachzentrierten Ansatz und die Transparenz durch die Sprachmittel von ExpL, Workflow-Konzepte und die teilweise Technologieunabhängigkeit des ExpL-Workflow-Systems. Mit ExpL liegt eine auf die Bedürfnisse von Experimentatoren zugeschnittene Beschreibungssprache vor, mit der Experimentier-Workflows unabhängig von einer spezifischen Technologie (z. B. einem Simulationssystem) definiert werden können. Somit lassen sich die Beschreibungen von Modell und Modellexperiment trennen. Dadurch wird die Beschreibung der Experimente wiederverwendbar und besser verständlich. Dies erhöht die Transparenz von Experimentier-Workflows und ermöglicht den Austausch mit anderen Wissenschaftlern, die sich dadurch auf wissenschaftliche Fragestellungen und nicht auf technologische konzentrieren können.

13. Weiterführende Arbeiten

Das ExpL-Wf-System ist mit seiner Flexibilität darauf ausgelegt, dass es in verschiedene Richtungen weiter entwickelt werden kann. In den folgenden Abschnitten kann daher nur jeweils ein Denkanstoß gegeben werden, wie sich das ExpL-Wf-System für eine bestimmte Thematik weiter entwickeln ließe.

Integration zusätzlicher Auswerteverfahren

ExpL verfügt mit dem Sprachelement `EvaluationPlan` (siehe Abschnitt 8.3.4) über eine Möglichkeit, verschiedene Auswerteverfahren auf eine Menge von Ergebnissen aus Experimentläufen anzuwenden. Allerdings wurde bisher nur ein sehr spezielles Auswerteverfahren in Form der GIS-DSL angebunden (und in der SLEUTH-Fallstudie angewendet, siehe Abschnitt 7.1). Auswerteverfahren sind jedoch essentiell, um mit großen Datenmengen aus einer hohen Anzahl von Experimentläufen umzugehen. Deshalb ist eine Erweiterung des ExpL-Wf-Systems um zusätzliche Auswerteverfahren sehr sinnvoll. Solche Auswerteverfahren können beispielsweise statistischer oder grafischer Natur sein. Zur Bereitstellung statistischer Funktionen könnte z. B. eine Anbindung an das Statistikpaket *R* geschaffen werden.

Grafische Auswerteverfahren sind, sofern es sich nicht um die reine Darstellung statistischer Größen handelt, abhängig von der Anwendungsdomäne. Beispielsweise ist das Werkzeug ClickWatch [SZS12a] in der Lage, Experimentdaten grafisch aufzubereiten, die in einem drahtlosen Maschennetzwerk auf der Basis des Click-Frameworks entstanden sind (das Click-Framework wird z. B. im HWL eingesetzt). Gerade die flexible Anpassung an eine Anwendungsdomäne ist eine Stärke das ExpL-Wf-Systems (z. B. durch den sprachzentrierten Ansatz).

Debugging und Pausieren von ExpL-Workflow

ExpL-Wf-Definitionen werden mittels Transformationen auf MWE-Workflow-Definitionen abgebildet und ausgeführt (siehe Abschnitt 9.3.1). MWE erlaubt weder das Pausieren der Ausführung von Workflow-Instanzen, noch das Inspizieren ihres Zustandes. Beides sind neben dem Modifizieren von Speicherinhalten typische Aktivitäten eines Debuggers, mit dem Fehler in der Workflow-Definition oder seiner Ausführung aufgespürt werden könnten. Das ExpL-Workflow-System bietet weder einen Debugger, noch die Möglichkeit, die Ausführung einer ExpL-Wf-Instanz anzuhalten und später fortzusetzen. Der ExpL-Baumeditor erlaubt es jedoch bereits, die Ausführung einer ExpL-Wf-Instanz zu verfolgen, indem Aktivitätsinstanzen als ausgeführt markiert werden. Diese Möglichkeit basiert auf dem Client-Server-Modell,

nach dem ExpL-Baumeditor und Experiment-Repositorium arbeiten. Dadurch werden Änderungen von ExpL-Wf-Instanz im Repositorium unmittelbar und automatisch sichtbar. Dieser Mechanismus kann auch umgekehrt genutzt werden, so dass Aktivitätsinstanzen über den ExpL-Baumeditor zur Ausführung gesperrt werden könnten, wodurch eine aktuell laufende ExpL-Wf-Instanz (kontrolliert) gestoppt werden könnte.

Um einen Debugger für ExpL-Wf-Definitionen zu realisieren, wären tiefgreifende, konzeptionelle Veränderungen am ExpL-Wf-System notwendig, die ohne eingehende Erforschung nicht ad hoc genannt werden können. Das Hauptproblem dabei ist es, Debugging-Informationen, die auf der Laufzeitebene der MWE-Workflow-Instanz gewonnen werden, zurück auf die Ebene der ExpL-Wf-Instanz zu beziehen. Nur dadurch kann der Experimentator jedoch einen Zusammenhang der Debugging-Informationen zu der von ihm erstellten ExpL-Wf-Definition herstellen. Für diesen Zweck ist ungünstigerweise die Transformation von ExpL-Wf-Instanz zu MWE-Workflow-Instanz nur unidirektional definiert. Zudem handelt es sich dabei um eine M2C-Transformation, d. h. für die Rückrichtung wäre ein Parser notwendig, was einen Bruch in der Technologie bedeutete (die Arbeit von Blunk [Blu09] scheint deshalb nicht anwendbar zu sein).

Unterstützung von Grid-Computing

Sofern es die Art der Modellexperimente (und des Modells) zulässt und notwendig macht, um innerhalb praktikabler Zeiträume Ergebnisse zu erreichen, führt man diese Modellexperimente parallel auf verschiedenen Computersystemen aus. Um die vorhandenen Hardware-Ressourcen effektiv auszunutzen, muss man einen höheren Planungsaufwand betreiben, insbesondere, wenn mehrere Experimentatoren die selben Hardware-Ressourcen nutzen möchten. Die Planung der Verteilung und Ausführung von Aktivitätsinstanzen auf Hardware-Ressourcen wird *Scheduling* genannt. Das ExpL-Wf-System erlaubt nur ein statisches Scheduling zur Entwurfszeit der ExpL-Wf-Definition, bei dem alle Zuweisungen von Ressourcen an Aktivitäten vom Experimentator manuell vorgenommen werden. Zudem bestimmt der Experimentator auch manuell zur Entwurfszeit der ExpL-Wf-Definition, welche Aktivitäten parallel ausgeführt werden können.

Die statische Kontrollfluss-Perspektive des Workflow-Definitionsmodells erlaubt es, zur Entwurfszeit eine Analyse des Aktivitäten-Transitionen-Graphen vorzunehmen (siehe Abschnitt 5.3.2). Anhand dessen kann entschieden werden, wann Aktivitäten eines Workflows parallel ausgeführt werden können. Es ist möglich, eine Kontrollfluss-Perspektive für eine ExpL-Wf-Definition zu erzeugen, weil die dafür notwendigen Informationen bereits in der ExpL-Wf-Definition enthalten sind.

Anhänge

Keine noch so große Zahl von Experimenten kann beweisen, daß ich recht habe; ein einziges Experiment kann beweisen, daß ich unrecht habe.

Albert Einstein

A. Sprachbeschreibung von ExpL in Form des Sprachmetamodells

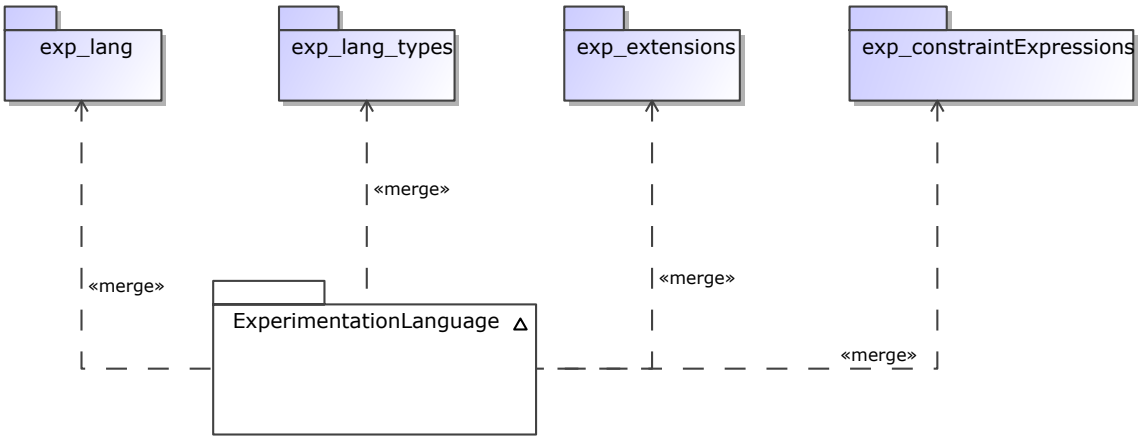


Abbildung A.1.: Paketstruktur von ExpL

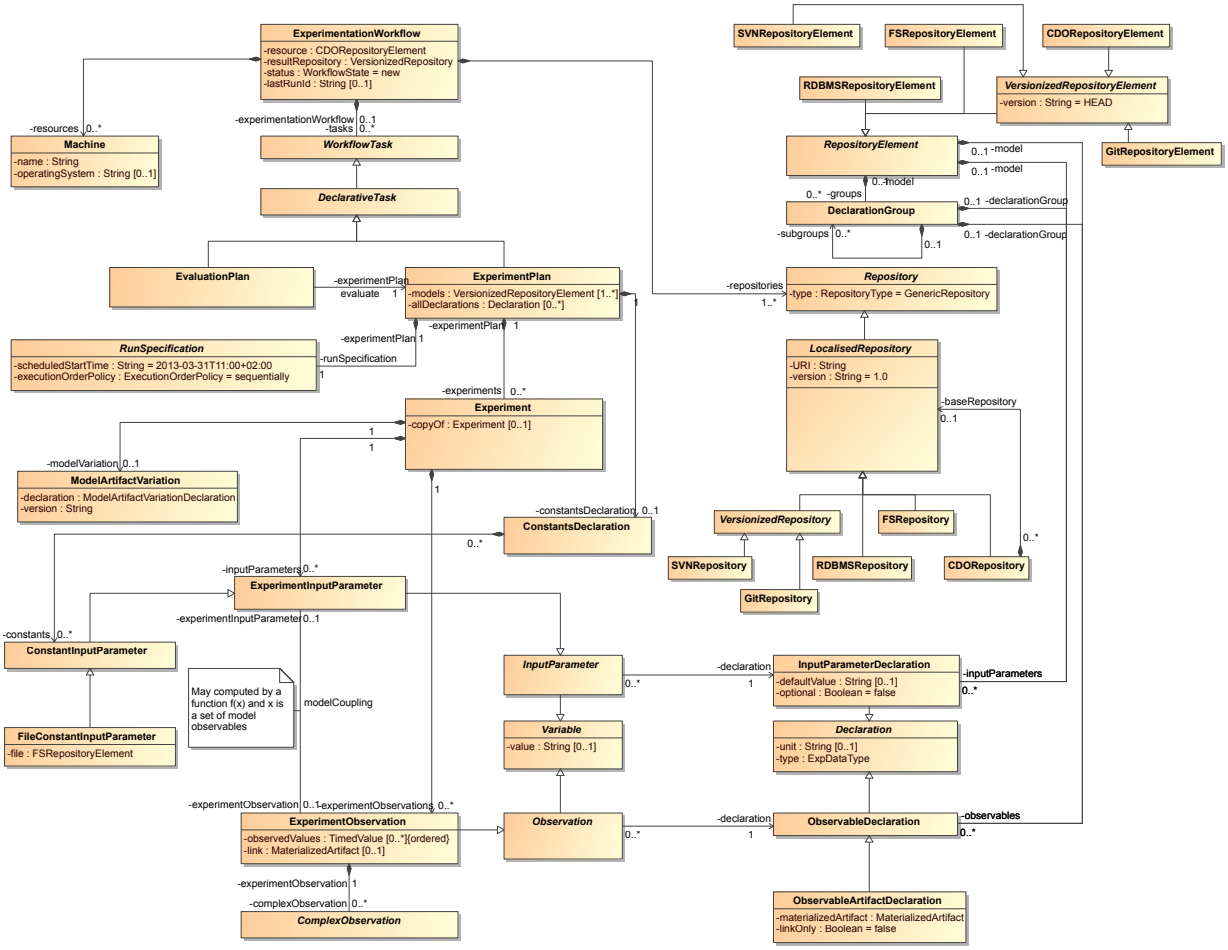


Abbildung A.2.: Kernklassen von ExpL im Paket `exp_lang`

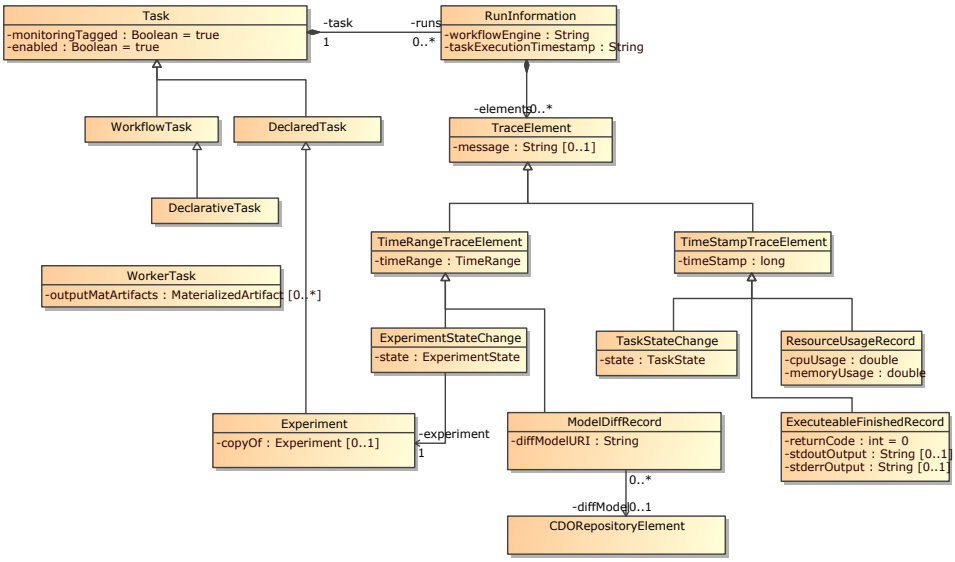


Abbildung A.3.: Sprachelement `RunInformation` von ExpL im Paket `exp_lang`

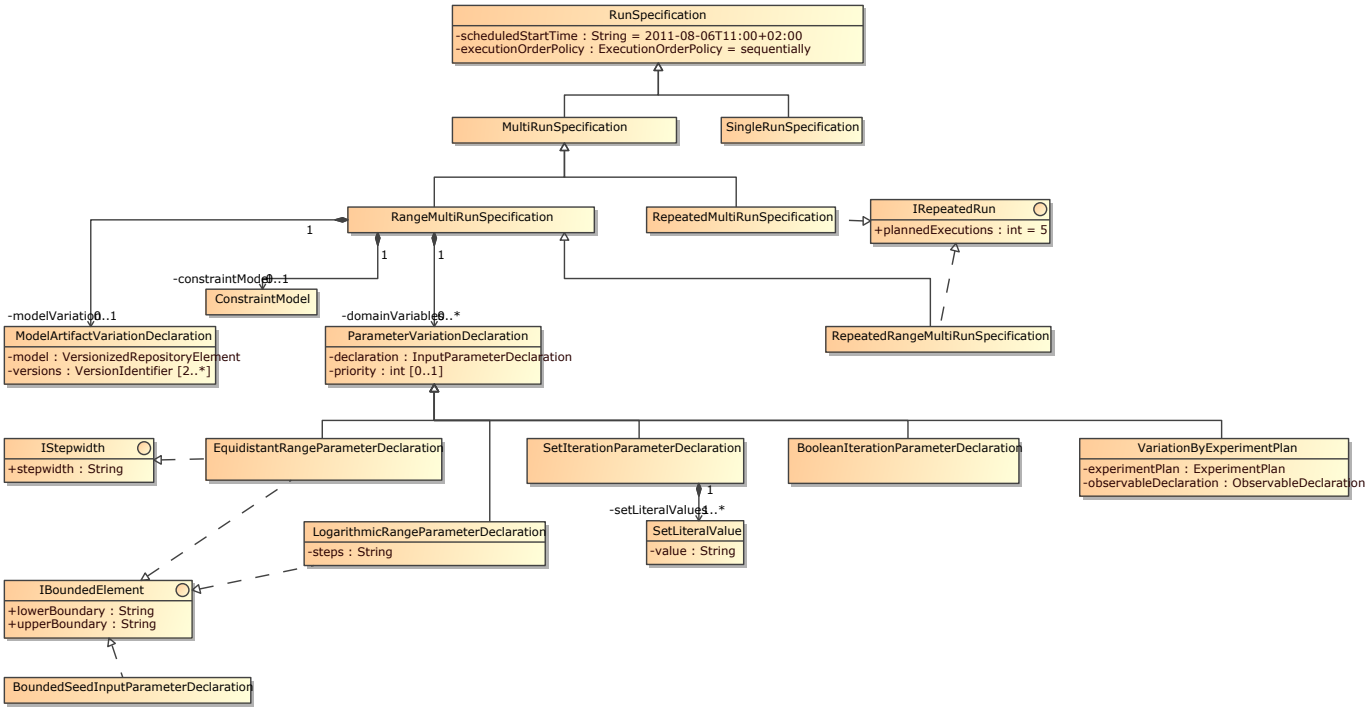


Abbildung A.4.: Sprachelement `RunSpecification` von ExpL im Paket `exp_lang`

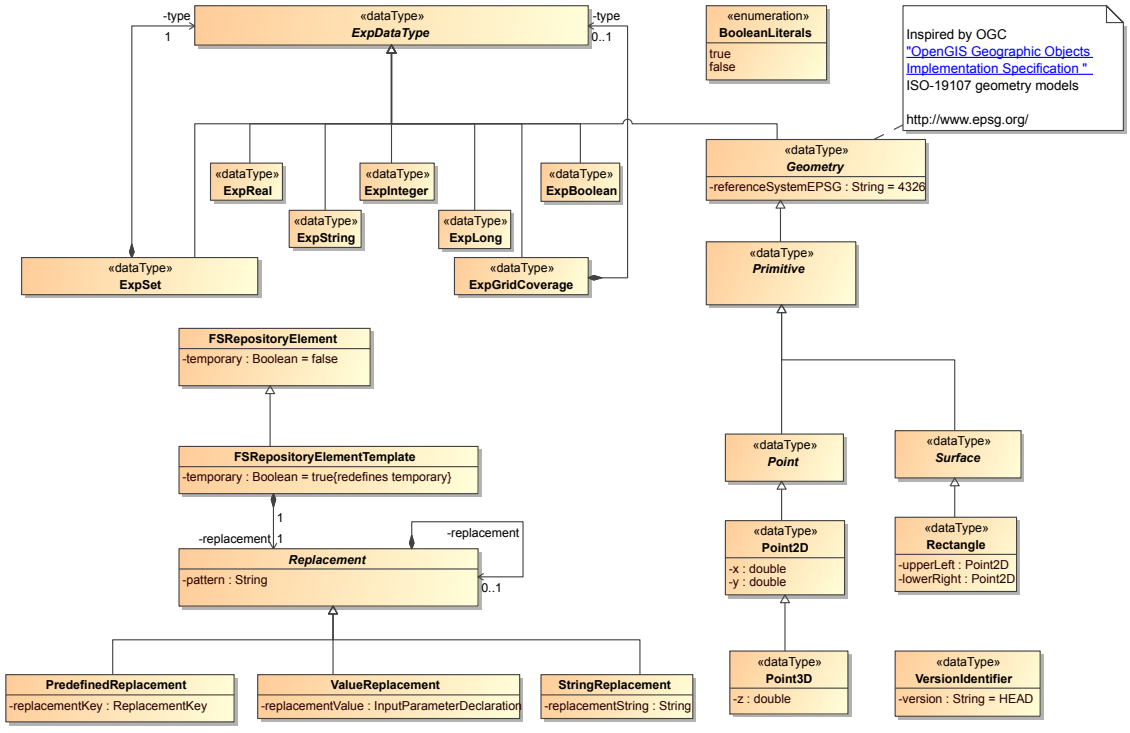


Abbildung A.5.: Klassen von ExpL des statischen Typsystems im Paket `exp_lang_types`

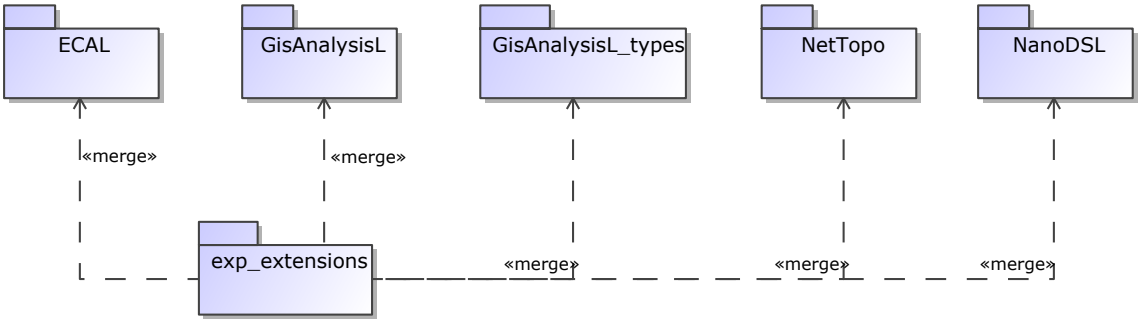


Abbildung A.6.: Paketstruktur von ExpL im Unterpaket `exp_extensions` für Konfigurations- und Auswertungs-DSL

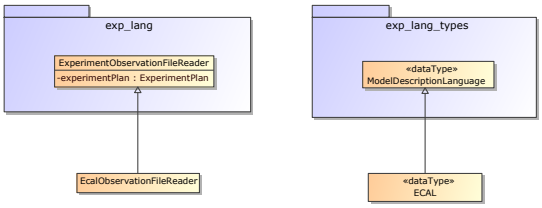


Abbildung A.7.: Klassen von ExpL im Unterpaket `ECAL`

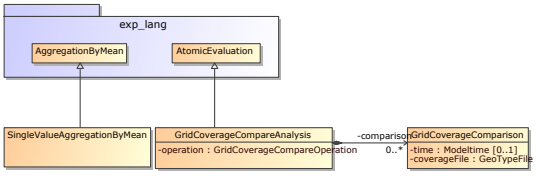


Abbildung A.8.: Klassen von ExpL im Unterpaket `GisAnalysisL`

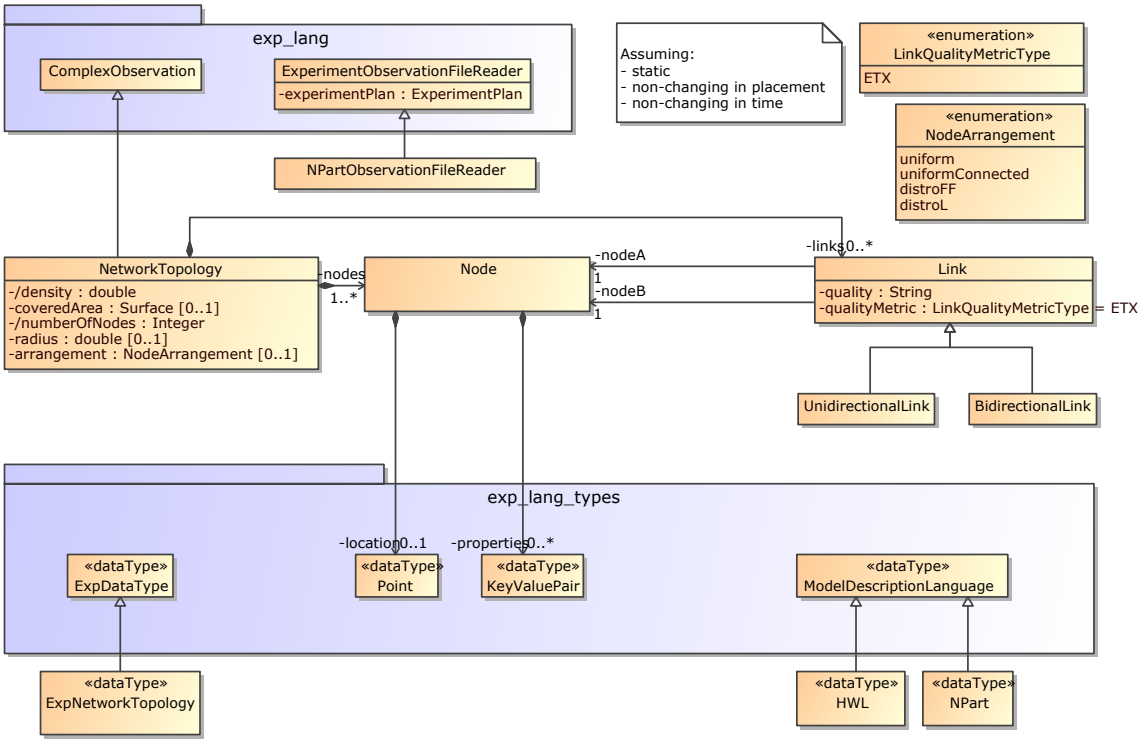


Abbildung A.9.: Klassen von ExpL im Unterpaket `NetTopo`

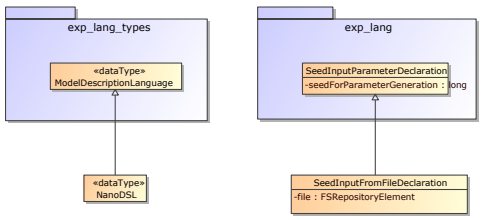


Abbildung A.10.: Klassen von ExpL im Unterpaket `NanoDSL`

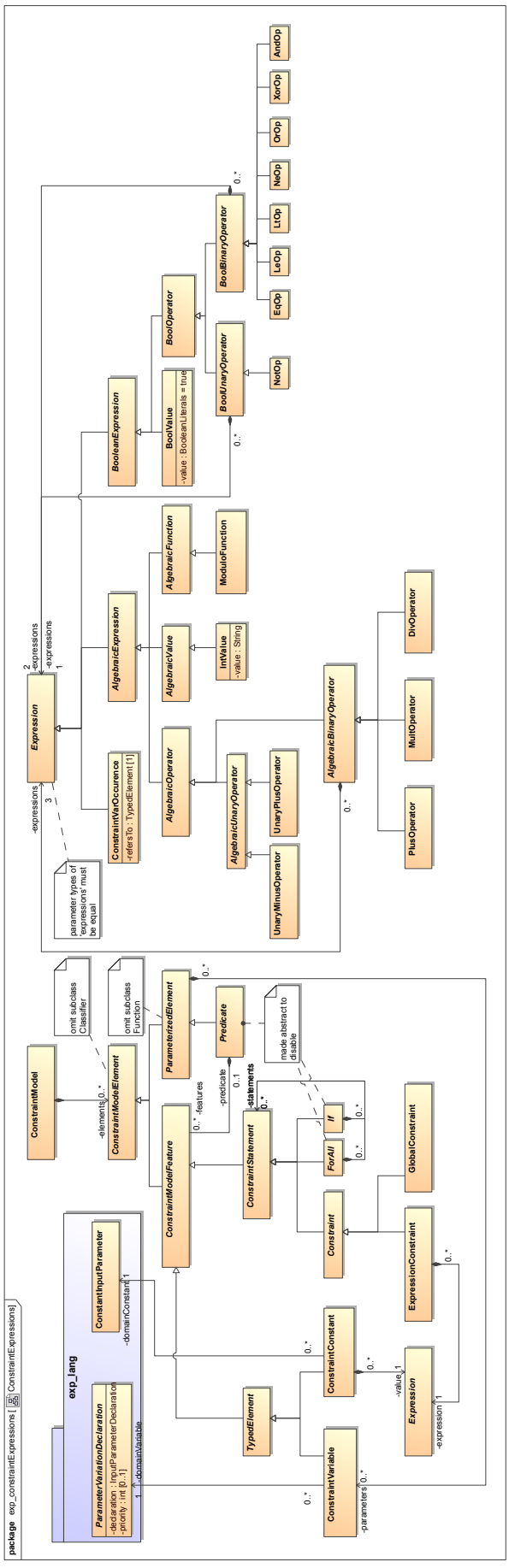


Abbildung A.11.: Klassen von ExpL im Unterpaket `exp_constraintExpressions`

B. Notationssprache von ExpL

B.1. Grammatik

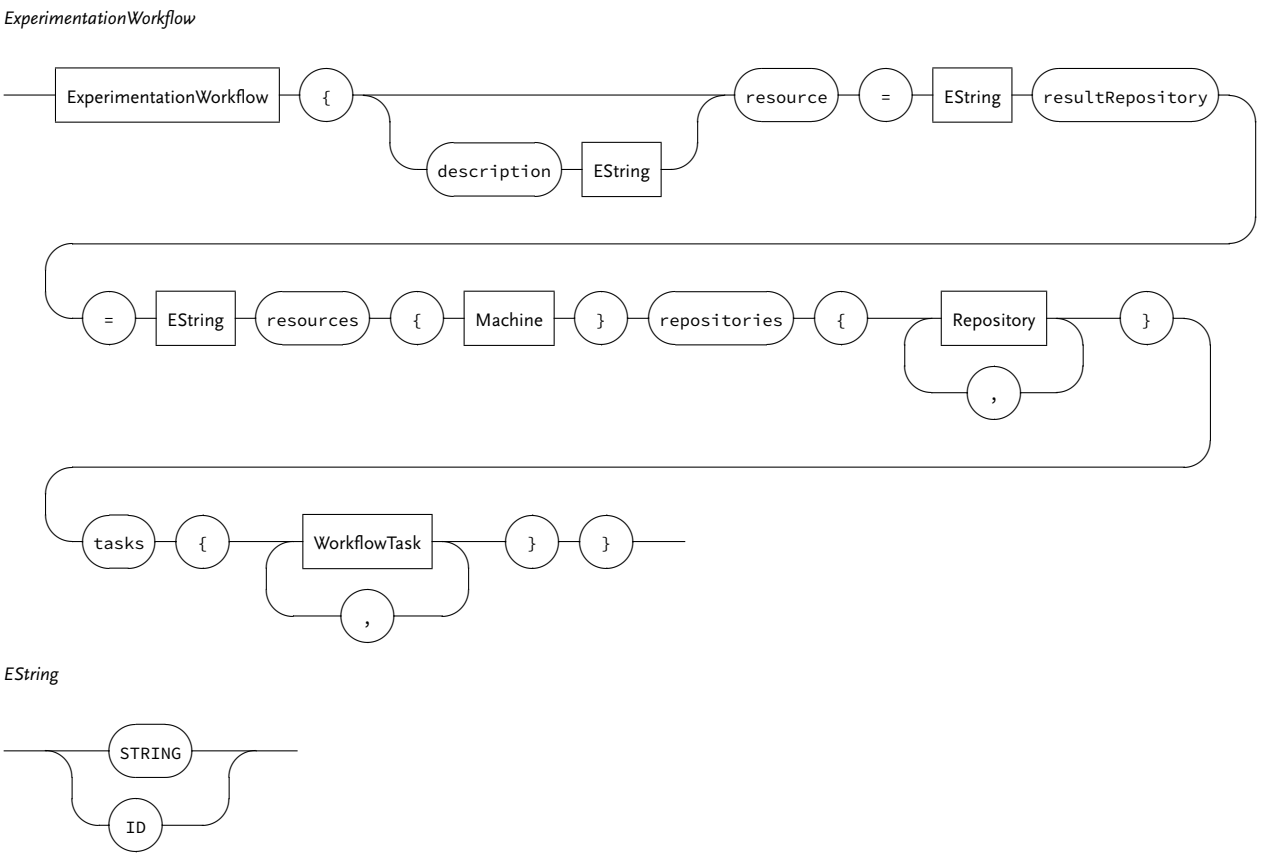


Abbildung B.1.: Auszug der kontextfreien Grammatik der Notationssprache von ExpL in Form eines Railroad-Diagramms

```
grammar expl.texteditor.Expl with org.eclipse.xtext.common.Terminals
// ExpL-Sprachmetamodell
import "platform:/resource/expl.metamodel.EMF/model/used/ExperimentationLanguage.ecore"
import "http://www.eclipse.org/emf/2002/Ecore" as ecore

ExperimentationWorkflow returns ExperimentationWorkflow:
 'ExperimentationWorkflow' '{'
   //Listing of Attributes
   (('description' description=EString)?
  &(('status' '=' status=WorkflowState)?
) &(('lastRunId' lastRunId=EString)?
))
   //Listing of Cross References
   ('resource' '=' resource=[CDORepositoryElement|EString]
  &('resultRepository' '=' resultRepository=[VersionizedRepository|EString]
))
   //Listing of Containment References
   ('resources' '{' resources+=Machine ( "," resources+=Machine)* '}'
  &('repositories' '{' repositories+=Repository ( "," repositories+=Repository)* '}' )
  &(('tasks' '{' tasks+=WorkflowTask ( "," tasks+=WorkflowTask)* '}' )?
))
   '}';

WorkflowTask returns WorkflowTask:
  EvaluationPlan | ExperimentPlan | AntScriptTask | GisAnalysisObservationFileReader |
      EcalObservationFileReader | NPartObservationFileReader | TupleFileReader |
      JavaExecutor | ODEMxExecutor | InputParameterFileWriter | PythonExecutor |
      ParallelTasksContainer | SequentialTasksContainer | GenericExecutor;

ExperimentPlan returns ExperimentPlan:
 'ExperimentPlan' name=EString ('for' models+=[VersionizedRepositoryElement|EString] (
     "," models+=[VersionizedRepositoryElement|EString])*
  &(('allDeclarations' '(' allDeclarations+=[Declaration|EString] ( ","
      allDeclarations+=[Declaration|EString])* ')' )?
))
 '{'
   //Listing of Attributes
   (('monitoringTagged' '=' monitoringTagged=EBoolean)?
  &(('enabled' enabled=EBoolean)?
) &(('description' description=EString)?
))
   //Listing of Containment References
   (('runs' '{' runs+=RunInformation ( "," runs+=RunInformation)* '}' )?
  &(runSpecification=RunSpecification
) &(('experiments' '{' experiments+=Experiment ( "," experiments+=Experiment)* '}' )?
) &(('constantsDeclaration' constantsDeclaration=ConstantsDeclaration)?
))
   '}';
```

Listing B.1: Auszug der Grammatik der Notationssprache von ExpL im Xtext-Format auf Basis der metamodellbasierte Sprachbeschreibung von ExpL

B.2. Bossel-Fischfang-Beispiel

```
GitRepository modelRepository {
 ModelRepository URI = /* ... */ workingDirectory = ModelRepositoryLocalCache
 elements {
  GitModelRepositoryElement "FISHFANG.MOD" {
   modelName "FISCHFANG-DYNAMIK" inputParameters {
    InputParameterDeclaration Fische as ExpReal
     { default "5000.0", unit "t" },
    InputParameterDeclaration Boote as ExpInteger { default "100" },
    InputParameterDeclaration Modellzeitstart as ExpReal
     { default "0.0", unit "Jahr" },
    InputParameterDeclaration Modellzeitstop as ExpReal
     { default "20.0", unit "Jahr" },
    InputParameterDeclaration Modellzeitschrittweite as ExpReal
     { default "0.02", unit "Jahr" },
    InputParameterDeclaration Fangchance as ExpReal { default "0.8" }
   }
   groups {
    DeclarationGroup Festparameter {
     inputParameters {
      InputParameterDeclaration Fanggebiet as ExpReal
       { default "100.0", unit "km2" },
      InputParameterDeclaration spezFischkapazitaet as ExpReal
       { default "100.0", unit "t/km2" },
      InputParameterDeclaration maxFischkapazitaet as ExpReal
       { unit "t" }
     }
    },
    DeclarationGroup Systemparameter {
     inputParameters {
      InputParameterDeclaration maxSpezFischzuwachsrate as ExpReal
       { default "1.0", unit "1/Jahr" },
      InputParameterDeclaration spezUnterhaltskosten as ExpReal
       { default "50000.0", unit "DM/(Boot.Jahr)" },
      InputParameterDeclaration BootsNeukosten as ExpReal
       { default "100000.0", unit "DM/Boot" },
      InputParameterDeclaration Bootslebensdauer as ExpReal
       { default "15.0", unit "Jahr" } /* 'Jahr' = '365 d'; SI 'd' == Tag*/
     }
    },
    DeclarationGroup Szenarioparameter {
     inputParameters {
      InputParameterDeclaration maxSpezFangmenge as ExpReal
       { default "100.0", unit "t Fisch/(Boot.Jahr)" },
      InputParameterDeclaration Fischpreis as ExpReal
       { default "1000.0", unit "DM/t Fisch" },
      InputParameterDeclaration Investitionsanteil_Boote as ExpReal
       { default "0.5" }
     }
    }
   }
```

```
51      observables {
          ObservableDeclaration Boote as ExpReal, /* Zeitreihe möglich */
53        ObservableDeclaration Fische as ExpReal, /* Zeitreihe möglich */
          ObservableDeclaration Fangmenge as ExpReal /* Zeitreihe möglich */
55      }
        language SIMPAS
57    }
    }
59  }
```

Listing B.2: Vollständige, textuell notierte MUX-Deklaration am Beispiel Bossel-Fischfang [Bos94, S. 169 ff.]

B.3. NetTopo-Beispiel

Das NetTopo-Beispiel ist im ExpL-Wf-System implementiert und zeigt eine sequentielle Modellkopplung über zwei Experimentpläne, die nacheinander ausgeführt werden (siehe Abbildung B.2 auf Seite 182). Im ersten Experimentplan `NetTopoGeneration` (siehe Listing B.4) werden Netzwerktopologien synthetisch durch den existierenden Generator NPart [MM09] erzeugt. Im zweiten Experimentplan `NetworkSimulation` (siehe Listing B.5 auf Seite 180) werden diese Netzwerktopologien als Eingaben für einen (fiktiven) Netzwerksimulator verwendet.

```
1   SVNRepository modelRepository {
      ModelRepository URI = /* ... */
3     workingDirectory = ModelRepositoryLocalCache
      elements {
5       SVNModelRepositoryElement "npart.topo" {
          modelName "npart test" inputParameters {
7           InputParameterDeclaration nodeArrangement as ExpSet { type ExpString },
            InputParameterDeclaration ^numberOfNodes as ExpInteger,
9           InputParameterDeclaration retries as ExpInteger,
            InputParameterDeclaration penalty as ExpInteger,
11          InputParameterDeclaration weight as ExpInteger,
            InputParameterDeclaration count as ExpInteger,
13          InputParameterDeclaration reduction as ExpInteger,
            InputParameterDeclaration outputFileType as ExpSet { type ExpString }
15        }
          groups {
17          DeclarationGroup Geometry {
              inputParameters {
19              InputParameterDeclaration xDimension as ExpInteger,
                InputParameterDeclaration yDimension as ExpInteger,
21              InputParameterDeclaration radius as ExpInteger
              }
23          }
          }
25        observables {
            ObservableArtifactDeclaration npart_outputFile {
```

```
27          artifact wd_exp_id/distribution0.fileext as ExpNetworkTopology
          }
29        }
        language NPart
31      }
    }
33  }
Legende: Schlüsselwort, Identifier, Referenz, "String", /* Kommentar */
```

Listing B.3: MUX-Deklaration im NetTopo-Beispiel für die Erzeugung einer synthetischen Netzwerktopologie

```
ExperimentPlan NetTopoGeneration for npart.topo {
 RangeMultiRunSpecification {
  parallel executionEnvironment clean {
4   supportedParallelActivities 5 binary JavaExecutor NPart {
     workingDirectory wd_exp_id parameters {
6      "--nodes" = numberOfNodesVariation,
       "--arrange" = nodeArrangementValues,
8      "--xDimension" = xDimensionConstant,
       "--yDimension" = yDimensionConstant,
10     "--radius" = radiusVariation,
       "--retries" = retriesConstant,
12     "--penalty" = penaltyConstant,
       "--secondaryWgh" = weightConstant,
14     "--count" = countConstant,
       "--reduction" = reductionConstant,
16     "--output" = outputFileTypes usage
     }
18   executable Executeable { file npart.jar }
    }
20  machine experimentserver.informatik.hu-berlin.de
  }
22 domainVariables {
    SetIteration nodeArrangementValues {
24    declaration nodeArrangement using {
       SetLiteralValue { value uniform },
26     SetLiteralValue { value uniformConnected },
       SetLiteralValue { value distroFF },
28     SetLiteralValue { value distroL }
      }
30  },
    EquidistantRange numberOfNodesVariation {
32    vary ^numberOfNodes "50" -> "100" # "50"
    },
34  EquidistantRange radiusVariation {
      vary ^radius "50" -> "100" # "50"
36  },
    SetIteration outputFileTypes {
38    declaration outputFileType using {
       SetLiteralValue { value dot },
40     SetLiteralValue { value ns2 }
```

```
      }
    }
  }
  constraintModel ConstraintModel {
    elements {
      ConstraintVariable cv_numberOfNodesVariation {
        domainVariable numberOfNodesVariation
      },
      ConstraintVariable cv_radiusVariation {
        domainVariable radiusVariation
      },
      ExpressionConstraint { /* radiusVariation < numberOfNodesVariation */
        expression LtOp { /* Operator '<' */
          expressions {
            ConstraintVarOccurence { refersTo cv_radiusVariation },
            ConstraintVarOccurence { refersTo cv_numberOfNodesVariation }
          }
        }
      }
    }
  }
  constantsDeclaration { constants { /* */ } }
 }
}
```

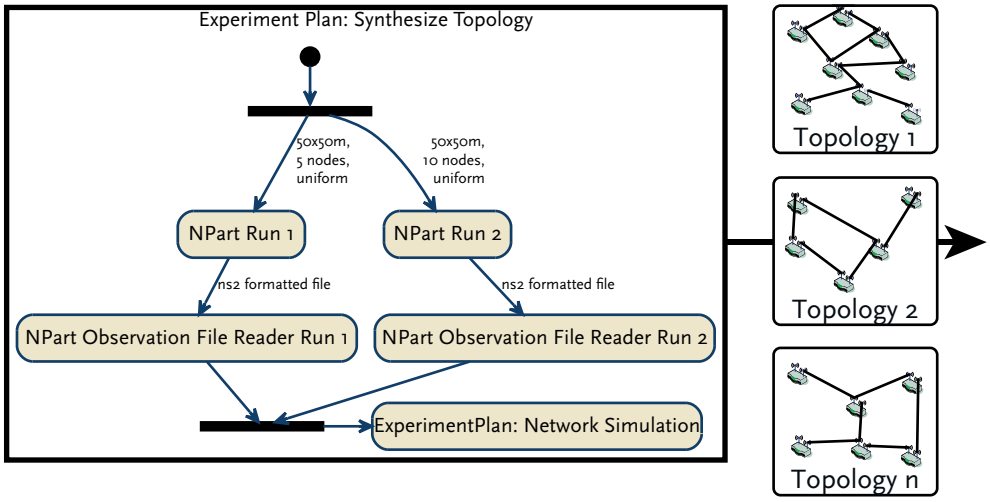
Listing B.4: Textuelle Repräsentation eines Experimentplanes in ExpL zur Beschreibung der Erzeugung einer synthetischen Netzwerktopologie

```
ExperimentPlan NetworkSimulation for MyRoutingProtocol.java {
 RangeMultiRunSpecification startAt "2011-09-05T11:00:00+02:00" parallel {
  EquidistantRange varyBitRate {
    vary bitRate 1 -> 10 # 1
  }
  SetIteration varyRadio {
    vary radioModel { "802.11bg", "802.11n" }
  }
  VariationByExperimentPlan varyTopology {
    using NetTopoGeneration /* dargestellt in Listing B.4 auf Seite 179 */
  }
  ConstraintModel {/*...*/}
 }
 executionEnvironment {
  supportedParallelActivities = 3 binary JavaExecutor "Simulator" {
    parameters {
      "--bitRate" = varyBitRate /* referenziert Deklaration in Zeile 3 */
      "--radio" = varyRadio /* referenziert Deklaration in Zeile 6 */
      "--topologyFile" = varyTopology /* referenziert Deklaration in Zeile 9 */
    }
    executable Executeable { file MyRoutingProtocol.jar }
  }
  machine experimentserver.informatik.hu-berlin.de
 }
```

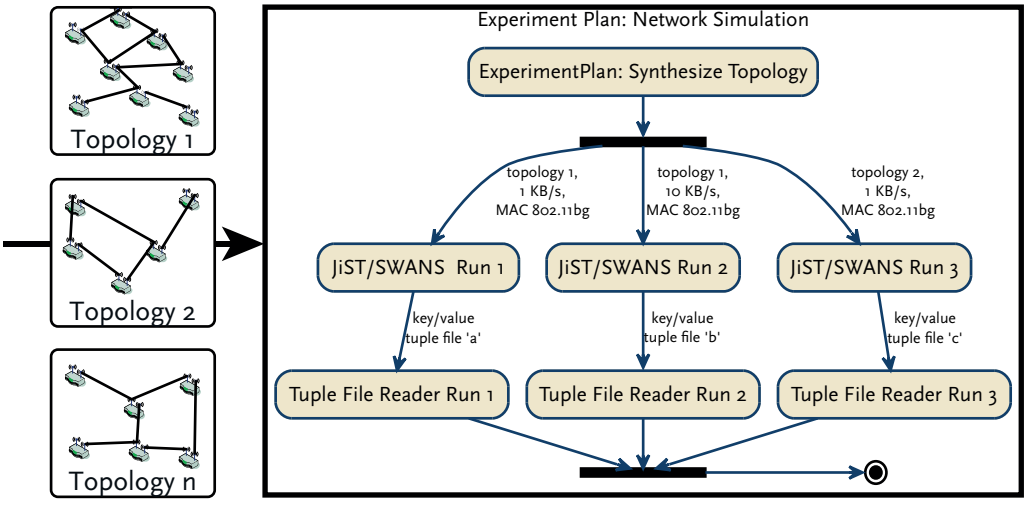
```
}

Legende: Schlüsselwort, Identifier, Referenz, "String", /* Kommentar */
```

Listing B.5: Textuelle Repräsentation eines Experimentplanes in ExpL zur Beschreibung von Netzwerksimulationen



(a) NetTopo-Beispiel: Visualisierung des Experimentplans `NetTopoGeneration`



(b) NetTopo-Beispiel: Visualisierung des Experimentplans `NetworkSimulation`

Abbildung B.2.: NetTopo-Beispiel mit zwei sequentiell ausgeführten Experimentplänen

C. Implementierungen der Fallstudien

C.1. SLEUTH-Fallstudie

```
ExperimentationWorkflow {
  description "SLEUTH*(P) Kalibrierungsphase"
  resource = SLEUTH*(P)_Kalibrierung /* referenziert Deklaration in Zeile 12 */
  resultRepository = ^ResultRepository
  resources {
    Machine "experimentserver.informatik.hu-berlin.de" { /* ... */ }
  } // Ende resources

  repositories {
    CDORepository experimentRepositorium {
      ConfigurationRepository URI = "tcp://experimentserver.informatik.hu-berlin.de:2036"
      elements { CDORepositoryElement SLEUTH*(P)_Kalibrierung { } }
    },
    SVNRepository ^ResultRepository { /* ... */ },
    FSRepository ResultRepositoryLocalCache { /* ... */ },
    SVNRepository SLEUTH*(P)-Repository { /* dargestellt in Listing
        C.2 auf Seite 184 */ },
    FSRepository ModelRepositoryLocalCache { /* ... */ },
    FSRepository ^SimulatorRepository {/* ... */ },
    FSRepository cainputdata {/* ... */ },
    FSRepository LibraryRepositoryLocalCache {/* ... */ },
    FSRepository validationdata {/* ... */ }
  } // Ende repositories
  tasks {
    InputParameterFileWriter PropertiesFileWriter for SLEUTH*(P)-Kalibrierung {
        produces ( exp_id/ID_exp_id.properties ) },
    ExperimentPlan SLEUTH*(P)-Kalibrierung for SLEUTH_Cal_Coarse_constCoeff0.ecal {
      /* dargestellt in Listing 10.1 auf Seite 146 */
    },
    EcalObservationFileReader EcalOutputFileReader for SLEUTH*(P)-Kalibrierung { },
    EvaluationPlan "SLEUTH*(P)-Kalibrierung GIS-Analyse" for SLEUTH*(P)-Kalibrierung {
      /* dargestellt in Listing C.4 auf Seite 187 */
    },
    EvaluationPlan "SLEUTH*(P)-Kalibrierung GIS-Analyse Aggregieren" for
        SLEUTH*(P)-Kalibrierung {
      /* dargestellt in Listing C.4 auf Seite 187 */
    },
    GisAnalysisObservationFileReader ImporterEvaluationPlan_mcGis for
        SLEUTH*(P)-Kalibrierung GIS-Analyse { },
    GisAnalysisObservationFileReader ImporterEvaluationPlan_mcGisAggregate for
        SLEUTH*(P)-Kalibrierung GIS-Analyse Aggregieren { }
```

```
  } // Ende tasks
}

Legende: Schlüsselwort, Identifier, Referenz, "String", /* Kommentar */
```

Listing C.1: Textuelle Repräsentation einer SLEUTH*(P)-Workflow-Definition in ExpL

```
SVNRepository SLEUTH*(P)-Repository {
  ModelRepository URI =
      "svn+ssh://experimentator@experimentserver.informatik.hu-berlin.de
      /repo/svn/ecal_project/model"
  workingDirectory = ModelRepositoryLocalCache
  elements {
    SVNModelRepositoryElement "SLEUTH_Cal_Coarse_constCoeff0.ecal" {
      modelName SLEUTH* inputParameters {
        InputParameterDeclaration DispersionCoeff as ExpReal { default "1.0" },
        InputParameterDeclaration BreedCoeff as ExpReal { default "1.0" },
        InputParameterDeclaration SpreadCoeff as ExpReal { default "1.0" },
        InputParameterDeclaration SlopeCoeff as ExpReal { default "1.0" },
        InputParameterDeclaration RoadCoeff as ExpReal { default "1.0" },
        InputParameterDeclaration Slope as ExpGridCoverage,
        InputParameterDeclaration Excluded as ExpGridCoverage,
        InputParameterDeclaration Urban as ExpGridCoverage,
        InputParameterDeclaration Transportation as ExpGridCoverage,
        InputParameterDeclaration CRITICAL_HIGH as ExpReal,
        InputParameterDeclaration CRITICAL_LOW as ExpReal,
        InputParameterDeclaration Seed as ExpReal { default "1" },
        InputParameterDeclaration BOOM as ExpReal,
        InputParameterDeclaration BUST as ExpReal,
        InputParameterDeclaration SlopeSensitivity as ExpReal,
        InputParameterDeclaration RoadSensitivity as ExpReal,
        InputParameterDeclaration MIN_SLOPE_RESISTANCE as ExpReal,
        InputParameterDeclaration MIN_ROAD_GRAVITY as ExpReal,
        InputParameterDeclaration MIN_DIFFUSION as ExpReal,
        InputParameterDeclaration MIN_SPREAD as ExpReal,
        InputParameterDeclaration MIN_BREED as ExpReal,
        InputParameterDeclaration MAX_SLOPE_RESISTANCE as ExpReal,
        InputParameterDeclaration MAX_ROAD_GRAVITY as ExpReal,
        InputParameterDeclaration MAX_DIFFUSION as ExpReal,
        InputParameterDeclaration MAX_SPREAD as ExpReal,
        InputParameterDeclaration MAX_BREED as ExpReal,
        InputParameterDeclaration CRITICAL_SLOPE as ExpReal,
        InputParameterDeclaration MAX_ROAD_VALUE as ExpReal
      }
      observables {
        ObservableArtifactDeclaration breedcoeff {
          artifact exp_id/observation/observable/observable as ExpReal
        },
        ObservableArtifactDeclaration dispersioncoeff {
          artifact exp_id/observation/observable/observable as ExpReal
        },
        ObservableArtifactDeclaration spreadcoeff {
```

```
44          artifact exp_id/observation/observable/observable as ExpReal
        },
46      ObservableArtifactDeclaration slopecoeff {
          artifact exp_id/observation/observable/observable as ExpReal
48      },
        ObservableArtifactDeclaration roadcoeff {
50        artifact exp_id/observation/observable/observable as ExpReal
        },
52      ObservableArtifactDeclaration growthrate {
          artifact exp_id/observation/observable/observable as ExpReal
54      },
        linkOnly ObservableArtifactDeclaration obs_urban_0 {
56        artifact exp_id/observation/obs_urban/observable_d0.tif as ExpGridCoverage
        },
58      linkOnly ObservableArtifactDeclaration obs_urban_1 {
          artifact exp_id/observation/obs_urban/observable_d0.tif as ExpGridCoverage
60      },
        linkOnly ObservableArtifactDeclaration obs_urban_2 {
62        artifact exp_id/observation/obs_urban/observable_d0.tif as ExpGridCoverage
        },
64      linkOnly ObservableArtifactDeclaration obs_urban_3 {
          artifact exp_id/observation/obs_urban/observable_d0.tif as ExpGridCoverage
66      },
        linkOnly ObservableArtifactDeclaration obs_urban_4 {
68        artifact exp_id/observation/obs_urban/observable_d0.tif as ExpGridCoverage
        },
70      linkOnly ObservableArtifactDeclaration obs_urban_5 {
          artifact exp_id/observation/obs_urban/observable_d0.tif as ExpGridCoverage
72      },
        linkOnly ObservableArtifactDeclaration obs_urban_6 {
74        artifact exp_id/observation/obs_urban/observable_d0.tif as ExpGridCoverage
        },
76      linkOnly ObservableArtifactDeclaration obs_urban_7 {
          artifact exp_id/observation/obs_urban/observable_d0.tif as ExpGridCoverage
78      },
        linkOnly ObservableArtifactDeclaration obs_urban_8 {
80        artifact exp_id/observation/obs_urban/observable_d0.tif as ExpGridCoverage
        },
82      linkOnly ObservableArtifactDeclaration obs_urban_9 {
          artifact exp_id/observation/obs_urban/observable_d0.tif as ExpGridCoverage
84      },
        linkOnly ObservableArtifactDeclaration obs_urban_10 {
86        artifact exp_id/observation/obs_urban/observable_d0.tif as ExpGridCoverage
        },
88      linkOnly ObservableArtifactDeclaration obs_urban_11 {
          artifact exp_id/observation/obs_urban/observable_d0.tif as ExpGridCoverage
90      },
        linkOnly ObservableArtifactDeclaration obs_urban_12 {
92        artifact exp_id/observation/obs_urban/observable_d0.tif as ExpGridCoverage
        },
94      linkOnly ObservableArtifactDeclaration obs_urban_13 {
          artifact exp_id/observation/obs_urban/observable_d0.tif as ExpGridCoverage
```

```
96        },
          linkOnly ObservableArtifactDeclaration obs_urban_14 {
98          artifact exp_id/observation/obs_urban/observable_d0.tif as ExpGridCoverage
          },
100       linkOnly ObservableArtifactDeclaration obs_urban_15 {
            artifact exp_id/observation/obs_urban/observable_d0.tif as ExpGridCoverage
102       },
          linkOnly ObservableArtifactDeclaration obs_urban_16 {
104         artifact exp_id/observation/obs_urban/observable_d0.tif as ExpGridCoverage
          },
106       linkOnly ObservableArtifactDeclaration obs_urban_17 {
            artifact exp_id/observation/obs_urban/observable_d0.tif as ExpGridCoverage
108       },
          linkOnly ObservableArtifactDeclaration obs_urban_18 {
110         artifact exp_id/observation/obs_urban/observable_d0.tif as ExpGridCoverage
          },
112       linkOnly ObservableArtifactDeclaration obs_urban_19 {
            artifact exp_id/observation/obs_urban/observable_d0.tif as ExpGridCoverage
114       }
        }
116   },
      SVNModelRepositoryElement "SLEUTH_Cal_Coarse_constCoeff0.ecal" {
118     modelName RoadsModelGen inputParameters {
        InputParameterDeclaration t0 as ExpGridCoverage,
120     InputParameterDeclaration t1 as ExpGridCoverage
      }
122   }
    }
124 }

126 Legende: Schlüsselwort, Identifier, Referenz, "String", /* Kommentar */
```

Listing C.2: MUX-Deklaration des SLEUTH*-Modells als Element eines `SVNRepository` im Workflow der SLEUTH-Fallstudie

```
executionEnvironment {
2   binary JavaExecutor SLEUTHEcaExecutor {
      vmargs {
4       "-Xms2048m",
        "-Xmx4096m"
6     }
      classpath "" parameters {
8       "paramfile=" = exp_id/ID_exp_id.properties,
        "outputdir=" = exp_id/observation
10    }
      executable Executeable {
12      file "eca.SLEUTHSim.class" dependency {
          ExecutionDependency "java jar libraries" {
14          artifacts ( eca.jar, jdisco.jar, geotools-2.5.7/*, JAI_1.1.3/* )
          }
16      }
      }
```

```
 }
 machine experimentserver.informatik.hu-berlin.de
}

Legende: Schlüsselwort, Identifier, Referenz, "String", /* Kommentar */
```

Listing C.3: ExpL-Sprachelement `ExecutionEnvironment`, das die Ausführungsumgebung im Workflow der SLEUTH-Fallstudie beschreibt

```
EvaluationPlan "SLEUTH*(P)-Kalibrierung GIS-Analyse" for SLEUTH*(P)-Kalibrierung {
 declarations {
  GridCoverageCompareAnalysis LeeSaleeSingleRun {
   input (obs_urban_0, obs_urban_1, obs_urban_10, obs_urban_11, obs_urban_12,
       obs_urban_13, obs_urban_14, obs_urban_15, obs_urban_16, obs_urban_17,
       obs_urban_18, obs_urban_19, obs_urban_2, obs_urban_3, obs_urban_4,
       obs_urban_5, obs_urban_6, obs_urban_7, obs_urban_8, obs_urban_9) in
       analysis/exp_id/analysis_name/plan_id/analysis_name.dat as ExpReal
   comparison {
    GridCoverageComparison {
     coverageFile = tirana/urb00_2.tif modeltime "12"
    },
    GridCoverageComparison {
     coverageFile = tirana/urb07_2.tif modeltime "19"
    }
   }
  },
  GridCoverageCompareAnalysis OASingleRun {
   operation OVERALLACCURACY
   input (obs_urban_0, obs_urban_1, obs_urban_10, obs_urban_11, obs_urban_12,
       obs_urban_13, obs_urban_14, obs_urban_15, obs_urban_16, obs_urban_17,
       obs_urban_18, obs_urban_19, obs_urban_2, obs_urban_3, obs_urban_4,
       obs_urban_5, obs_urban_6, obs_urban_7, obs_urban_8, obs_urban_9) in
       analysis/exp_id/analysis_name/plan_id/analysis_name.dat as ExpReal
   comparison {
    GridCoverageComparison {
     coverageFile = tirana/urb00_2.tif modeltime "12"
    },
    GridCoverageComparison {
     coverageFile = tirana/urb07_2.tif modeltime "19"
    }
   }
  },
  GridCoverageCompareAnalysis PASingleRun {
   operation PRODUCERACCURACY
   input (obs_urban_0, obs_urban_1, obs_urban_10, obs_urban_11, obs_urban_12,
       obs_urban_13, obs_urban_14, obs_urban_15, obs_urban_16, obs_urban_17,
       obs_urban_18, obs_urban_19, obs_urban_2, obs_urban_3, obs_urban_4,
       obs_urban_5, obs_urban_6, obs_urban_7, obs_urban_8, obs_urban_9) in
       analysis/exp_id/analysis_name/plan_id/analysis_name.dat as ExpReal
   comparison {
    GridCoverageComparison {
     coverageFile = tirana/urb00_2.tif modeltime "12"
```

```
    },
    GridCoverageComparison {
      coverageFile = tirana/urb07_2.tif modeltime "19"
    }
  }
},
GridCoverageCompareAnalysis PANotSingleRun {
  operation PRODUCERACCURACYNEG
  input (obs_urban_0, obs_urban_1, obs_urban_10, obs_urban_11, obs_urban_12,
      obs_urban_13, obs_urban_14, obs_urban_15, obs_urban_16, obs_urban_17,
      obs_urban_18, obs_urban_19, obs_urban_2, obs_urban_3, obs_urban_4,
      obs_urban_5, obs_urban_6, obs_urban_7, obs_urban_8, obs_urban_9) in
      analysis/exp_id/analysis_name/plan_id/analysis_name.dat as ExpReal
  comparison {
    GridCoverageComparison {
      coverageFile = tirana/urb00_2.tif modeltime "12"
    },
    GridCoverageComparison {
      coverageFile = tirana/urb07_2.tif modeltime "19"
    }
  }
},
GridCoverageCompareAnalysis UASingleRun {
  operation USERACCURACY
  input (obs_urban_0, obs_urban_1, obs_urban_10, obs_urban_11, obs_urban_12,
      obs_urban_13, obs_urban_14, obs_urban_15, obs_urban_16, obs_urban_17,
      obs_urban_18, obs_urban_19, obs_urban_2, obs_urban_3, obs_urban_4,
      obs_urban_5, obs_urban_6, obs_urban_7, obs_urban_8, obs_urban_9) in
      analysis/exp_id/analysis_name/plan_id/analysis_name.dat as ExpReal
  comparison {
    GridCoverageComparison {
      coverageFile = tirana/urb00_2.tif modeltime "12"
    },
    GridCoverageComparison {
      coverageFile = tirana/urb07_2.tif modeltime "19"
    }
  }
},
GridCoverageCompareAnalysis UANotSingleRun {
  operation USERACCURACYNEG
  input (obs_urban_0, obs_urban_1, obs_urban_10, obs_urban_11, obs_urban_12,
      obs_urban_13, obs_urban_14, obs_urban_15, obs_urban_16, obs_urban_17,
      obs_urban_18, obs_urban_19, obs_urban_2, obs_urban_3, obs_urban_4,
      obs_urban_5, obs_urban_6, obs_urban_7, obs_urban_8, obs_urban_9) in
      analysis/exp_id/analysis_name/plan_id/analysis_name.dat as ExpReal
  comparison {
    GridCoverageComparison {
      coverageFile = tirana/urb00_2.tif modeltime "12"
    },
    GridCoverageComparison {
      coverageFile = tirana/urb07_2.tif modeltime "19"
    }
```

```
72      }
    },
74    GridCoverageCompareAnalysis KappaSingleRun {
      operation KAPPA
76      input (obs_urban_0, obs_urban_1, obs_urban_10, obs_urban_11, obs_urban_12,
            obs_urban_13, obs_urban_14, obs_urban_15, obs_urban_16, obs_urban_17,
            obs_urban_18, obs_urban_19, obs_urban_2, obs_urban_3, obs_urban_4,
            obs_urban_5, obs_urban_6, obs_urban_7, obs_urban_8, obs_urban_9) in
            analysis/exp_id/analysis_name/plan_id/analysis_name.dat as ExpReal
      comparison {
78      GridCoverageComparison {
          coverageFile = tirana/urb00_2.tif modeltime "12"
80      },
        GridCoverageComparison {
82        coverageFile = tirana/urb07_2.tif modeltime "19"
        }
84    }
    }
86  }
},
88 EvaluationPlan "SLEUTH*(P)-Kalibrierung GIS-Analyse Aggregieren" for
      SLEUTH*(P)-Kalibrierung {
  declarations {
90  SingleValueAggregationByMean LeeSaleeMean {
      preAnalysis = LeeSaleeSingleRun in analysis/exp_id/analysis_name/analysis_name.dat
          as ExpReal /* referenziert Deklaration in Zeile 3*/
92  },
    SingleValueAggregationByMean ProducerAccuracyMean {
94    preAnalysis = PASingleRun in analysis/exp_id/analysis_name/analysis_name.dat as
          ExpReal /* referenziert Deklaration in Zeile 26*/
    },
96  SingleValueAggregationByMean ProducerAccuracyNotMean {
      preAnalysis = PANotSingleRun in analysis/exp_id/analysis_name/analysis_name.dat as
          ExpReal /* referenziert Deklaration in Zeile 38*/
98  },
    SingleValueAggregationByMean UserAccuracyMean {
100   preAnalysis = UASingleRun in analysis/exp_id/analysis_name/analysis_name.dat as
          ExpReal /* referenziert Deklaration in Zeile 50*/
    },
102 SingleValueAggregationByMean UserAccuracyNotMean {
      preAnalysis = UANotSingleRun in analysis/exp_id/analysis_name/analysis_name.dat as
          ExpReal /* referenziert Deklaration in Zeile 62*/
104 },
    SingleValueAggregationByMean KappaMean {
106   preAnalysis = KappaSingleRun in analysis/exp_id/analysis_name/analysis_name.dat as
          ExpReal /* referenziert Deklaration in Zeile 74*/
    },
108 SingleValueAggregationByMean OAMean {
      preAnalysis = OASingleRun in analysis/exp_id/analysis_name/analysis_name.dat as
          ExpReal /* referenziert Deklaration in Zeile 14*/
110 }
  }
```

```
}

Legende: Schlüsselwort, Identifier, Referenz, "String", /* Kommentar */
```

Listing C.4: ExpL-Sprachelement `EvaluationPlan`, das die GIS-DSL-basierte Auswertung im Workflow der SLEUTH-Fallstudie beschreibt

C.2. Nano-Fallstudie

```
SETUP {
  val r = 0.45
  val a = 250
  val s = 0.3
  project_name = "L3_t70_a250_r45_s0.3"
}

UNIT {
  length_unit = nanometers
  time_unit = femtoseconds
  frequency_unit = terahertz
  loss_unit = dB per cm
}

STRUCTURE {
  material GaP = 3.3

  Slab {
    material = GaP
    position = (0,0,0)
    thickness = 70
  }

  Lattice{
    lattice_type = hexagonal
    lattice_size = (19,19)
    lattice_distance = a
    hole_radius_to_distance = 0.30
    hole_radius = r * a

    delete horizontal line from center with span (5)
    move (4,0) {
      x_offset = s
      y_offset = 0
    }
    move (-4,0) {
      x_offset = -1*s
      y_offset = 0
    }
  }
```

```
}

SIMULATION {
  simulation_time = 500
  background_index = 1
  dx = 1/20
  dy = 1/20
  dz = 1/20
  boundary_x_min = Symmetric
  boundary_y_min = Anti-Symmetric
  boundary_z_min = Symmetric
}

SOURCES {
  Electric {
    position = (10,10,10)
    dipole_theta = 90
    dipole_phi = 90
    Broadband {
      wavelength_center = 640
      wavelength_span = 200
    }
  }
}

MONITORS {
  Spec spec {
    position= (0,0,0)
  }
}
```

Listing C.5: Beispielmodell zur Beschreibung einer optischen Nanostruktur und einer darauf basierenden Simulation in der textuellen Repräsentation der Nano-DSL

```
clear;

#----------------------------
# Project Name
#----------------------------
project_name ="Tanabe";

#----------------------------
# Structure
#----------------------------

#layer parameter
  slab_thickness = 59;  #in nm
  slab_material = "GaP";
  slab_z_pos = 0;
  slab_surface_roughness=0; #the slab will be coveres with cylinders on random position
    cover_ratio=0; #number_of_cyl * pi*r² / slab_surface
    roughness_cylinder_radius=100;
```

```
    roughness_cylinder_heigth=0;
    rand_data_file = "rand1.txt";
  metal_thickness = 10;
  metal_material = "gold";
  metal_z_pos = (slab_thickness+metal_thickness)/2;

#lattice parameters
  lattice_constant = 209; #in nm
  r_to_a = 0.28;
  hole_radius = r_to_a*lattice_constant;  #in nm

#size of structure
  layer_number_x = 19;  #must be an odd number
  layer_number_y = 17;  #must be an odd number

#random hole position
  random_hole = 0;  #either 0 (off) or 1 (on)
  sigma_pos = 0.25;   #standard deviation in nm
  sigma_size = 1.25;  #standard deviation in nm
  rand_data_file='rand_data3.txt';

#waveguide(s)   #waveguide in x direction. x,y center
  waveguide = [0,0,0,0.945];
  #waveguide = [0];
  #syntax: [x,y,length,width]
  #x,y,length,width in units of layers
  #length=0 for full crystal
  #multiple waveguides separated by semicolons
  #set [0] for none
  shift = 0;

#Tanabe cavity(ies)
  tanabe = [0,0,6,4,2,2];
  #syntax: [x,y,A,B,C,length of zone A]
  #multiple cavities separated by semicolons
  #set [0] for none

#Grating outcoupler(s)
  outcoupler = [0];
  #syntax: [x,y,frequency]
  #x,y in units of layers
  #frequency in THz
  #multiple couplers separated by semicolons
  #set [0] for none

#------------------------------
# Simulation
#------------------------------

#simulation parameters
  simulation_time = 250;  #in fs
  background_index = 1.0;
```

```
#mesh size
 dx = 1/20;   #in units of lattice constant
 dy = sqrt(3)*dx/1.7;  #in units of lattice constant
 dz = 1/20;   #in units of lattice constant

#boundaries
 boundary_x_min = "PML"; #use either 'PML'
 boundary_y_min = "PML"; # or 'Symmetric'
 boundary_z_min = "PML"; # or 'Anti-Symmetric'

#-----------------------------
# Sources
#-----------------------------

#choose either wavelength or time domain setting
 set_wavelength = 1;  #1 for broadband source, 0 otherwise
 wavelength_center = 637.4; #in nm
 wavelength_span = 100;  #in nm
 set_time_domain = 0;  #1 for narrowband source, 0 otherwise
 frequency = c/1000/wavelength_center;  #in THz
 pulse_length = 50;  #in fs
 pulse_offset = 100;  #in fs

#dipole source
 dipole_type = 0;  #0/1 for electric/magnetic dipole
 dipole_pos = [-4,0,0];  #[x,y,z] in units of layers
 dipole_theta = 90;  #in degree
 dipole_phi = 90;  #in degree

#-----------------------------
# Monitors
#-----------------------------

#time monitors
 timemon = [0.05,0.05,0];
 #syntax: [x,y,z]
 #x,y,z in units of layers
 #multiple monitors separated by semicolons
 #set [0] for none

#spectral monitors
 specmon = [.05,0.05,0];
 #syntax: [x,y,z]
 #x,y,z in units of layers (2D)
 #x,y, in units of ax/2,ay (3D)
 #z in nm
 #multiple monitors separated by semicolons
 #set [0] for none

#box monitors
 z_max_monitor = 1;
```

```
124 #-----------------------------
    # Build Structure
126 #-----------------------------
    addpath("scripts");
128 #make_structure2D;   #create 2D structure

130 runfile="run_L3rough.sh";
    write(runfile,"!/bin/bash\n");
132
    project_name = project_name+num2str(roughness_cylinder_heigth);
134 make_structure3D;   #create 3D structure
    runstr="\"/usr/local/lumerical/bin/../mpich2/nemesis/bin/mpiexec\" -n 4
        \"/usr/local/lumerical/bin/FDTD-par-mpich2nem\" \""+currentfilename+"\"";
136 write(runfile,runstr);

138 make_autocad_script;   #creates an autocad script to copy the structure
```

Listing C.6: Beispielmodell zur Beschreibung einer optischen Nanostruktur und einer darauf basierenden Simulation in der textuellen Syntax von FDTD Solutions

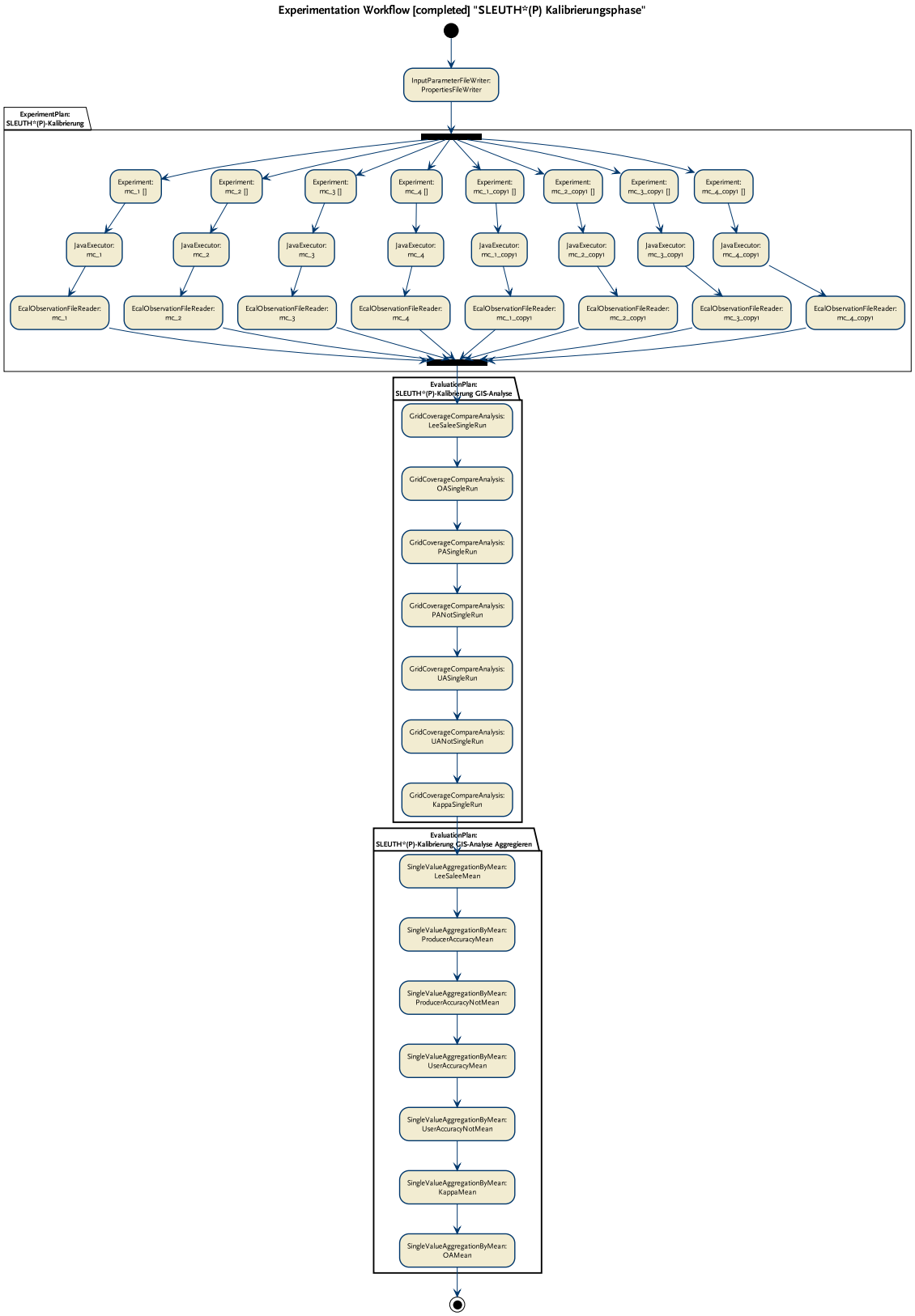


Abbildung C.1.: Durch den ExpL-Workflow-Viewer automatisch generierte, graphische Darstellung des Workflows aus der SLEUTH-Fallstudie in Listing C.1 auf Seite 183

D. Implementierungsaspekte von ExpL

D.1. Transformationen

```
// this is the first of two modules to execute an experimentation workflow
// 1. expl.wfe.ExecuteExperimentationWorkflow (internal)
// 2. expl.wfe.ExecuteExperimentPlan (generated by 1.)
module expl.wfe.ExecuteExperimentationWorkflow

import expl.wfe.mwe.TransactionalCdoRepositoryManager
import expl.wfe.mwe.ExperimentPlanMapper
import expl.wfe.mwe.FileDeleter
import expl.wfe.csp.CSPSolver

// default values, overwritten by ExpL-Wfe properties
var explModelSlot = "explModelSlot"
var cdoConnectionSlot = "cdoConnectionSlot_executeExpWf"
var cdoTransactionSlot = "cdoTransactionSlot_executeExpWf"
var explMetaModelFile =
    "platform:/resource/expl.metamodel.EMF/model/used/ExperimentationLanguage.ecore"
var explMetaModelFileExecutionTime = explMetaModelFile
var fileEncoding = "UTF-8"
var workflowRepositoryName = "configurationrepository"
var cdoServerConnectionString = "localhost:2036"
var tempDir = "C:/GitRepositories/expl-dist/wf"

// DO NOT delete, because it may be a git-repository location
var resultRepositoryLocalCache = "src/expl/wfe/temp"

var experimentPlanWfFilename = "ExecuteExperimentPlan.mwe2"
var flatzincFilename = "Constraints.fzn"
var variablesSolutionSlot = "variablesSolutionSlot"

// controlling the experimentation workflow
// (needs to be strings, because of globalVarDef)
var doMapping = "true"
var skipSimulator = "true"
var skipEvaluation = "false"
var removeOldObservations = "true"

Workflow {
  bean = org.eclipse.emf.mwe.utils.StandaloneSetup {
    uriMap = {
      from = "http:///ExperimentationLanguage.ecore"
```

```
    to = explMetaModelFile
  }
  registerGeneratedEPackage = "ExperimentationLanguage.ExperimentationLanguagePackage"
}
bean = org.eclipse.xtend.typesystem.emf.EmfMetaModel : explMetaModel {
  metaModelPackage = "ExperimentationLanguage.ExperimentationLanguagePackage"
}

// open connection and session
component = TransactionalCdoRepositoryManager {
  cdoServerConnectionString = cdoServerConnectionString
  workflowRepositoryName = workflowRepositoryName
  workflowName = "/${workflowName}"
  explModelSlot = explModelSlot
  cdoConnectionSlot = cdoConnectionSlot
  action = "OPEN_SESSION"
}

// open transaction, read from repository
component = TransactionalCdoRepositoryManager {
  cdoServerConnectionString = cdoServerConnectionString
  workflowRepositoryName = workflowRepositoryName
  workflowName = "/${workflowName}"
  explModelSlot = explModelSlot
  cdoConnectionSlot = cdoConnectionSlot
  cdoTransactionSlot = cdoTransactionSlot
  action = "OPEN_TRANSACTION"
}

component = org.eclipse.xtend.check.CheckComponent {
  metaModel = explMetaModel
    checkFile = "expl::wfe::ValidateExpLModel"
    emfAllChildrenSlot = explModelSlot
    abortOnError = false // otherwise there is no error text message!!!
//    warnIfNothingChecked = true
}

component = org.eclipse.emf.mwe.core.container.IfComponent {
  cond = doMapping

  component = FileDeleter {
    fileName = flatzincFilename
    path = tempDir
  }

  // generate flatzinc file for constraint solving
  component = org.eclipse.xpand2.Generator {
    metaModel = explMetaModel
    expand = "expl::wfe::CSPSolverTransformation::main('${flatzincFilename}') FOR
        ${explModelSlot}"
    outlet = {
      path = tempDir
```

```
    }
    dumpContext = true
    fileEncoding = fileEncoding
  }

  // solve constraints in flatzinc file by JaCoP constraint solver
  component = CSPSolver {
    // contains a list of values for input variables as all solutions for the
        constraints
    variablesSolutionSlot = variablesSolutionSlot
    flatzincFile = "${tempDir}/${flatzincFilename}"
  }

  component = ExperimentPlanMapper {
    explModelSlot = explModelSlot
    // contains a list of values for input variables as all solutions for the
        constraints
    variablesSolutionSlot = variablesSolutionSlot
    removeGeneratedExperiments = true
  }

  // open transaction, read from repository
  component = TransactionalCdoRepositoryManager {
    cdoTransactionSlot = cdoTransactionSlot
    workflowName = "/${workflowName}"
    action = "COMMIT_TRANSACTION"
  }

  component = org.eclipse.xtend.check.CheckComponent {
    metaModel = explMetaModel
      checkFile = "expl::wfe::PostMappingExpLModel"
      emfAllChildrenSlot = explModelSlot
      abortOnError = false // otherwise there is no error text message!!!
//      warnIfNothingChecked = true
  }
 }

 // generate sub-workflow MWE2 file ExecuteExperimentPlan
 component = org.eclipse.xpand2.Generator {
  metaModel = explMetaModel
  expand = "expl::wfe::ExperimentPlanExecutionWf::main('${experimentPlanWfFilename}',
      '${explMetaModelFileExecutionTime}') FOR ${explModelSlot}"
  outlet = {
    path = tempDir
  }
  dumpContext = true
  globalVarDef = {
    name = "skipSimulator"
    value = skipSimulator
  }
  globalVarDef = {
    name = "removeOldObservations"
```

```
    value = removeOldObservations
  }
  globalVarDef = {
    name = "skipEvaluation"
    value = skipEvaluation
  }
  fileEncoding = fileEncoding
 }

 // prepare possible input files for simulator run
 // should move to expl.wfe.ExecuteExperimentPlan
 component = org.eclipse.xpand2.Generator {
  metaModel = explMetaModel
  expand = "expl::wfe::ExperimentPlanPreparation::main FOR ${explModelSlot}"
  outlet = {
    path = resultRepositoryLocalCache
  }
  dumpContext = true
  fileEncoding = fileEncoding
 }

 // close transaction
 component = TransactionalCdoRepositoryManager {
  cdoConnectionSlot = cdoConnectionSlot
  workflowName = "/${workflowName}"
  action = "CLOSE_SESSION"
 }
} // end workflow
```

Listing D.1: Interner MWE-Workflow der ExpL-Wfe zur Erzeugung des MWE-Laufzeitworkflows

```
import ExperimentationLanguage; // Import des ExpL-Sprachmetamodells

// mit Xtend definierte Erweiterungen
extension org::eclipse::xtend::util::stdlib::naming;
extension expl::wfe::misc::ISO8601DateParser;
extension expl::wfe::misc::MiscJava;

/*
* Structural checks.
*/
context InputParameterDeclaration ERROR
 "The declaration of the input parameter '" + loc(this) + "' must reference a model
     directly or via DeclarationGroup." :
 model != null || declarationGroup != null;

context ObservableDeclaration ERROR
 "The declaration of the observable '" + loc(this) + "' must reference a model directly
     or via DeclarationGroup." :
 model != null || declarationGroup != null;
```

```
context FSRepository if type != RepositoryType::TemporaryRepository ERROR
 "The path of '" + qualifiedName(this) + "' does not exists: " + this.URI + ".":
 existsFile(fileUri2fileName(this.URI)); // mit Xtend definierte Erweiterung verwenden
// ...

/*
* Checks for supported types.
*/
context EquidistantRangeParameterDeclaration ERROR
 "EquidistantRangeParameterDeclaration '" + qualifiedName(this) + "' can only refer to
     declarations of type ExpInteger or ExpReal, but it refers to '" +
     qualifiedName(declaration) + "' of type " + this.declaration.type.metaType :
 this.declaration.type.metaType == ExpInteger || this.declaration.type.metaType ==
     ExpReal;
// ...

/*
* Checks for semantic constraints.
*/
context VariationByExperimentPlan ERROR
 "The VariationByExperimentPlan of experiment plan '" +
     ((RangeMultiRunSpecification)this.eContainer).experimentPlan + "' can not
     reference the experiment plan, which it belongs to." :
 this.experimentPlan != ((RangeMultiRunSpecification)this.eContainer).experimentPlan;

context VariationByExperimentPlan ERROR
 "The VariationByExperimentPlan of experiment plan '" +
     ((RangeMultiRunSpecification)this.eContainer).experimentPlan + "' can only
     reference an experiment plan before that one it belongs to." :
 ((ExperimentationWorkflow)this.eRootContainer).tasks.indexOf(this.experimentPlan) <
     ((ExperimentationWorkflow)this.eRootContainer).tasks.indexOf(
 ((RangeMultiRunSpecification)this.eContainer).experimentPlan);

context RunSpecification ERROR
 "The scheduled start time '" + scheduledStartTime + "' of '" + qualifiedName(this) +
     "' is not a valid ISO8601 date value: " + getISO8601DateErrorMessage() :
 isValidISO8601Date(scheduledStartTime);
// ...

/*
* Currently unsupported things.
*/
context EvaluationExpressionDeclaration ERROR
 "Language element EvaluationExpressionDeclaration '" + qualifiedName(this) + "' used
     in evaluation plan '" + this.evaluationPlan.name + "' is not supported yet." :
 false;

context ModelArtifactVariationDeclaration ERROR
  "The model reference of '" + qualifiedName(this) + "' has the type '" +
      this.model.metaType + "'. But currently only the types 'GitRepositoryElement' or
      'GitModelRepositoryElement' are supported." :
```

```
 this.model.metaType == GitRepositoryElement || this.model.metaType ==
     GitModelRepositoryElement;
// ...


/*
* Warnings.
*/
context SVNModelRepositoryElement WARNING
 "Model repository element '" + name + "' in '" + this.repository.name + "' should have
     at least one observable declared." :
 !this.observables.isEmpty;

context ExperimentationWorkflow WARNING
 "Experimentation workflow '" + this.resource.name + "' should have at least one task
     defined." :
 !this.tasks.isEmpty;
// ...
```

Listing D.2: Auszug des Check-Programms zur strukturellen und semantischen Überprüfung einer ExpL-Wf-Definition vor deren Instanziierung und Ausführung

D.2. Quellen von ExpL

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: ExpL Workflow Engine (Wfe)
Bundle-SymbolicName: expl.wfe;singleton:=true
Bundle-Version: 0.8.3.qualifier
Bundle-ClassPath: .,
 lib/args4j.jar,
 lib/eca.jar,
 lib/JaCoP.jar,
 lib/npart.jar
Bundle-Vendor: Frank Kühnlenz
Bundle-RequiredExecutionEnvironment: JavaSE-1.6
Export-Package: .,
 expl.wfe,
 expl.wfe.mwe
Require-Bundle: org.eclipse.core.runtime;bundle-version="3.5.0",
 org.eclipse.emf.cdo;bundle-version="4.0.1",
 org.eclipse.emf.mwe.core;bundle-version="1.2.1",
 org.eclipse.emf.mwe2.launch;bundle-version="2.1.0",
 org.eclipse.xtext;bundle-version="2.1.0",
 org.eclipse.xpand;bundle-version="1.2.0",
 org.eclipse.xtend;bundle-version="1.2.0",
 org.apache.ant;bundle-version="1.7.1",
 org.eclipse.emf.ecore.change;bundle-version="2.5.1",
 org.eclipse.emf;bundle-version="2.6.0",
```

```
26  ExperimentationLanguage.model;bundle-version="0.8.0",
    org.eclipse.xtend.util.stdlib;bundle-version="1.2.0",
28  expl.texteditor;bundle-version="0.8.1";resolution:=optional,
    org.eclipse.jgit;bundle-version="1.0.0",
30  org.eclipse.net4j;bundle-version="4.0.1",
    org.eclipse.net4j.db;bundle-version="4.0.0",
32  org.eclipse.net4j.tcp;bundle-version="4.0.0",
    org.eclipse.emf.cdo.net4j;bundle-version="4.0.1",
34  org.eclipse.xtend.typesystem.emf;bundle-version="1.2.0"
Bundle-ActivationPolicy: lazy
```

Listing D.3: Java-Manifest des Programms ExpL-Workflow-Engine, an dem Versionsabhängigkeiten zu anderen Programmen und Bibliotheken sichtbar werden

Name	Abschnitt	Eclipse-Paketname
ExpL-Sprachmetamodell	8.3 auf Seite 105	`expl.metamodel`
ExpL-Texteditor	9.3.3.2 auf Seite 131	`expl.texteditor`
ExpL-Viewer	9.3.4 auf Seite 132	`expl.eclipse.visualization`
ExpL-Workflow-Engine	9.3.5 auf Seite 133	`expl.wfe` `expl.eclipse.contribution`

Tabelle D.1.: Eclipse-Pakete der Bestandteile des ExpL-Wf-Systems

Literaturverzeichnis

[20006a] ; Object Management Group (OMG) (Veranst.): *Meta Object Facility (MOF) 2.0 Core Specification*. 2006

[20006b] *Seismic eArly warning For EuRope (SAFER)*. 06 2006. http://www.saferproject.net. – abgerufen 05.03.2013

[20007b] *OMG Unified Modeling Language, Infrastructure, V2.1.2*. 11 2007. http://www.omg.org/spec/UML/2.1.2/Infrastructure/PDF

[4] *DistSim*. 2008. https://sarwiki.informatik.hu-berlin.de/DistSim. – abgerufen 17.03.2013

[20009f] *PragmaDev RTDS*. 06 2009. http://www.pragmadev.com. – abgerufen 09.03.2013

[6] *Object Management Group (OMG)*. 10 2011. http://www.omg.org/. – abgerufen 05.03.2013

[20113a] *Modeling Workflow Engine 2 (MWE2)*. 02 2013. http://help.eclipse.org/helios/index.jsp?topic=%2Forg.eclipse.xtext.doc%2Fhelp%2FMWE2.html

[20113b] *PlantUML*. 03 2013. http://plantuml.sourceforge.net/. – abgerufen 05.03.2013

[AEF+09] Ahrens, Klaus ; Eveslage, Ingmar ; Fischer, Joachim ; Kühnlenz, Frank ; Weber, Dorian: The challenges of using SDL for the development of wireless sensor networks. In: Reed, Rick (Hrsg.) ; Bilgic, Attila (Hrsg.) ; Gotzhein, Reihnard (Hrsg.): *14th System Design Languages Forum* Bd. ISBN 978-3-624-04553-0. Springer, Bochum, Germany, 09 2009 (LNCS 5719), S. 200–221

[ant13] *Apache Ant*. 02 2013. http://ant.apache.org/. – abgerufen 05.03.2013

[Ash57] Ashby, William R.: *An introduction to cybernetics*. Chapman & Hall, London, UK, 1957. – 295 S.

[ATHKB03] Aalst, W.M.P. van d. ; Ter Hofstede, A.H.M. ; Kiepuszewski, B. ; Barros, A.P.: Workflow Patterns. In: *Distributed and Parallel Databases*, Bd. 14, Nr. 1, 2003, 5–51. http://library.tue.nl/csp/dare/LinkToRepository.csp?recordnumber=613310. – ISSN 0926-8782

[Bal09] Balzert, Helmut: *Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering*. Bd. 1. 3. Auflage. Spektrum Akademischer Verlag, 2009. – 624 S. – ISBN: 978-3827417053

[Bal11] Balzert, Helmut: *Lehrbuch der Softwaretechnik: Entwurf, Implementierung, Installation und Betrieb*. Bd. 2. 3. Auflage. Spektrum Akademischer Verlag, 2011. – 596 S. – ISBN: 978-3827417060

[Bar04] Barr, Rimon: *An efficient, unifying Approach to Simulation using Virtual Machines*. Cornell University, Dissertation, 05 2004

[BBEF11] Blunk, Andreas ; Brumbulli, Mihal ; Eveslage, Ingmar ; Fischer, Joachim: Modeling real-time applications for wireless sensor networks using standardized techniques. In: *SIMULTECH 2011*, 2011, S. 161–167

[Ber02] Berlin, Humboldt-Univeristät zu: *Satzung über die Grundsätze der Humboldt-Universität zu Berlin zur Sicherung guter wissenschaftlicher Praxis und über den Umgang mit Vorwürfen wissenschaftlichen Fehlverhaltens*. 07 2002. http://www.hu-berlin.de/forschung/administration/0702gsgw.pdf. – abgerufen 05.03.2013

[BF11] Brumbulli, Mihal ; Fischer, Joachim: SDL Code Generation for Network Simulators. In: Kraemer, Frank (Hrsg.) ; Herrmann, Peter (Hrsg.): *System Analysis and Modeling: About Models* Bd. 6598. Springer Berlin / Heidelberg, 2011. – ISBN 978–3–642–21651–0, S. 144–155

[BHVR05] Barr, Rimon ; Haas, Zygmunt J. ; Van Renesse, Robbert: Scalable wireless ad hoc Network Simulation. In: *Handbook on Theoretical and Algorithmic Aspects of Sensor, Ad hoc Wireless, and Peer-to-Peer Networks*, 2005, S. 297–311

[BKS+07] Barth, M. ; Kouba, J. ; Stingl, J. ; Löchel, B. ; Benson, O.: Modification of visible spontaneous emission with silicon nitride photonic crystal nanocavities. In: *Optics Express*, Bd. 15, Nr. 25, 12 2007, 17231–17240. http://dx.doi.org/10.1364/OE.15.017231. – DOI 10.1364/OE.15.017231

[Blu09] Blunk, Andreas: *MODEF - Ein generisches Debugging-Framework für domänenspezifische Sprachen mit metamodellbasierter Sprachdefinition auf der Basis von Eclipse, EMF und EProvide*. Humboldt-Universität zu Berlin, Diplomarbeit, 2009. http://www2.informatik.hu-berlin.de/Institut/struktur/systemanalyse/zweckverband/blunk-diplomarbeit-final-2009.pdf

[Bos94] Bossel, Hartmut: *Modellbildung und Simulation: Konzepte, Verfahren und Modelle zum Verhalten dynamischer Systeme*. 2., veränderte Auflage. Vieweg, Braunschweig/Wiesbaden, 1994. – 402 S. http://info.ub.hu-berlin.de/primus/001195562. – ISBN 3-528-15242-7

[Bry80] Bryant, Raymond M.: SIMPAS: A Simulation Language Based on Pascal / University of Wisconsin-Madison Department of Computer Sciences. 06 1980. http://digital.library.wisc.edu/1793/58220. 1980. – Forschungsbericht

[Byk72] Bykov, V. P.: Spontaneous Emission in a Periodic Structure. In: *Soviet Journal of Experimental and Theoretical Physics*, Bd. 35, 1972, S. 269. – ISSN 00385646

[Can02] Candau, Jeannette T.: *Temporal Calibration Sensitivity of the SLEUTH Urban Growth Model*. University of California, Santa Barbara, Diplomarbeit, 06 2002

[CDO13] *Connected Data Objects (CDO)*. 02 2013. http://wiki.eclipse.org/index.php?title=CDO&oldid=322490. – abgerufen 05.03.2013

[Cel91] Cellier, François E.: *Continuous System Modeling*. Springer-Verlag, New York, USA, 1991. – 755 S. – ISBN-13: 978-0387975023

[CH03] Czarnecki, Krzysztof ; Helsen, Simon: Classification of Model Transformation Approaches. In: *Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture* Bd. 45, 2003, S. 1–17

[CHG96] Clarke, Keith C. ; Hoppen, S. ; Gaydos, L.: Methods and techniques for rigorous calibration of a cellular automaton model of urban growth. In: *Third International Conference/Workshop on Integrating GIS and Environmental Modeling, Santa Fe, New Mexico*, 1996

[CHG97] Clarke, Keith C. ; Hoppen, S. ; Gaydos, L.: A self-modifying cellular automaton model of historical urbanization in the San Francisco Bay area. In: *Environment and Planning B: Planning and Design*, Bd. 24, Nr. 2, 1997, 247–261. http://dx.doi.org/10.1068/b240247. – DOI 10.1068/b240247

[Cla08] Clarke, Keith C. : *Kapitel* 3 A Decade of Cellular Urban Modeling with SLEUTH: Unresolved Issues and Problems. In: Brail, Richard K. (Hrsg.): *Planning Support Systems for Cities and Regions*. Lincoln Institute of Land Policy, Cambridge, MA, 2008, S. 47–60

[Cla11] Clarke, Keith C.: *SLEUTH-Anwendungen*. 11 2011. http://www.ncgia.ucsb.edu/projects/gig/v2/About/abApps.htm

[Dat98] ; ITU-T (Veranst.): *Data networks and open system communications, OSI networking and system aspects - Abstact Syntax Notation One (ASN.1); Information technology - ASN.1 Encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER), and Distinguished Encoding Rules (CER), ITU-T Recommendation X.690*. 1998

[DGST09] Deelman, Ewa ; Gannon, Dennis ; Shields, Matthew ; Taylor, Ian: Workflows and e-Science: An overview of workflow system features and capabilities. In: *Future Generation Computer Systems*, Bd. 25, Nr. 5, 2009, S. 528 – 540. http://dx.doi.org/10.1016/j.future.2008.06.012. – DOI 10.1016/j.future.2008.06.012. – ISSN 0167–739X

[DSS+05] Deelman, E. ; Singh, G. ; Su, M.H. ; Blythe, J. ; Gil, Y. ; Kesselman, C. ; Mehta, G. ; Vahi, K. ; Berriman, G.B. ; Good, J. u. a.: Pegasus: A framework for mapping complex scientific workflows onto distributed systems. In: *Scientific Programming*, Bd. 13, Nr. 3, 2005, S. 219–237

[Dud11] *Duden – Deutsches Universalwörterbuch*. 7. Bibliographisches Institut, Mannheim, 2011. – 2112 S. – ISBN 3411055065. – ISBN 978-3411055074

[EFO+03] Erdik, M. ; Fahjan, Y. ; Ozel, O. ; Alcik, H. ; Mert, A. ; Gul, M.: Istanbul Earthquake Rapid Response and the Early Warning System. In: *Bulletin of Earthquake Engineering*, Bd. 1, Nr. 01, 01 2003, 157-163. http://dx.doi.org/doi:10.1023/A:1024813612271. – DOI doi:10.1023/A:1024813612271

[EKF08] Efftinge, Sven ; Köhnlein, Jan ; Friese, Peter: Build your own textual DSL with Tools from the Eclipse Modeling Project / Eclipse Foundation. 04 2008. http://www.eclipse.org/articles/article.php?file=Article-BuildYourOwnDSL/index.html. 2008. – Forschungsbericht. – abgerufen 17.02.2013

[Eve08] Eveslage, Ingmar: *Simulation selbstorganisierender Erdbebenfrühwarnsysteme unter Einbindung eines Geoinformationssystems*. Humboldt-Universität zu Berlin, Diplomarbeit, 09 2008

[EZ11] Efftinge, Sven ; Zarnekow, Sebastian: Extending Java / Pragmatic Programmer Magazine. 12 2011. http://pragprog.com/magazines/2011-12/extending-java. 2011. – Forschungsbericht. – abgerufen 17.02.2013

[FA96] Fischer, Joachim ; Ahrens, Klaus: *Objektorientierte Prozeßsimulation in C++*. Addison-Wesley, 1996 (Reihe Praktische Informatik). – 360 S. http://info.ub.hu-berlin.de/primus/001180979. – ISBN 3-8273-1039-3

[FAWG07] Fischer, Joachim ; Ahrens, Klaus ; Witaszek, Dorota ; Gerstenberger, Ralf: *ODEMx – Object oriented Discrete Event Modelling*. 05 2007. http://www.sourceforge/projects/odemx

[Fis82] Fischer, Joachim: *Modellierung und Simulation Paralleler Diskreter, Kontinuierlicher und Kombinierter Prozesse in SIMULA*. Akad. der Wiss. der DDR, Zentrum für Rechentechnik, Berlin, 1982. – 170 S.

[FKAE09b] Fischer, Joachim ; Kühnlenz, Frank ; Ahrens, Klaus ; Eveslage, Ingmar: Model-based Development of Self-organizing Earthquake Early Warning Systems. In: *Simulation News Europe SNE*, Bd. 19, Nr. 3-4, 01 2009. http://www.asim-gi.org/index.php?id=19&no_cache=1&tx_pubdb_pi1[pnum]=3

[FKE+08a] Fischer, Joachim ; Kühnlenz, Frank ; Eveslage, Ingmar ; Ahrens, Klaus ; Nachtigall, Jens ; Lichtblau, Björn ; Heglmeier, Sebastian ; Milkereit, Claus ; Fleming, Kevin ; Picozzi, Matteo: Deliverable D4.21 Develop Software for Network Connectivity / Department of Computer Science, Humboldt-Universität zu Berlin and Section 2.1, GFZ Potsdam. 06 2008. http://www.saferproject.net/. 2008. – Forschungsbericht

[FKE+09c] Fischer, Joachim ; Kühnlenz, Frank ; Eveslage, Ingmar ; Ahrens, Klaus ; Nachtigall, Jens ; Lichtblau, Björn ; Milkereit, Claus ; Fleming, Kevin ;

Picozzi, Matteo: Deliverable D4.22 Middleware for Geographical Applications / Department of Computer Science, Humboldt-Universität zu Berlin and Section 2.1, GFZ Potsdam. 06 2009. http://www.saferproject.net/. 2009. – Forschungsbericht

[FKE+09d] Fischer, Joachim ; Kühnlenz, Frank ; Eveslage, Ingmar ; Ahrens, Klaus ; Nachtigall, Jens ; Lichtblau, Björn ; Milkereit, Claus ; Fleming, Kevin ; Picozzi, Matteo: Deliverable D4.26 Network Optimisation simulation for combined standard and low cost seismic network integration. Proposal for network optimisation / Department of Computer Science, Humboldt-Universität zu Berlin and Section 2.1, GFZ Potsdam. 07 2009. http://www.saferproject.net/. 2009. – Forschungsbericht

[FP10] Fowler, Martin ; Parsons, Rebecca: *Domain-specific Languages*. 1. Edition. Addison-Wesley Professional, 2010. – 640 S. http://martinfowler.com/bliki/DomainSpecificLanguage.html. – ISBN 978-0321712943

[FPD+07] Fahringer, Thomas ; Prodan, Radu ; Duan, Rubing ; Hofer, Jürgen ; Nadeem, Farrukh ; Nerieri, Francesco ; Podlipnig, Stefan ; Qin, Jun ; Siddiqui, Mumtaz ; Truong, Hong-Linh ; Villazon, Alex ; Wieczorek, Marek: ASKALON: A Development and Grid Computing Environment for Scientific Workflows. In: Taylor, Ian J. (Hrsg.) ; Deelman, Ewa (Hrsg.) ; Gannon, Dennis B. (Hrsg.) ; Shields, Matthew (Hrsg.): *Workflows for e-Science*. Springer London, 2007, S. 450–471. – ISBN 978-1-84628-519-6

[FPM+09] Fleming, Kevin ; Picozzi, Matteo ; Milkereit, Claus ; Kühnlenz, Frank ; Lichtblau, Björn ; Fischer, Joachim ; Zulfikar, C. ; Özel, O.: The Self-Organising Seismic Early Warning Information System (SOSEWIN). In: *Seismological Research Letters*, Bd. 80, Nr. 5, September/October 2009, S. 755 – 771. http://dx.doi.org/10.1785/gssrl.80.5.755. – DOI 10.1785/gssrl.80.5.755

[Gai79] Gaines, Brian R.: General Systems Research: Quo vadis. In: *Yearbook of the Society for General Systems Research* Bd. 24, General Systems, 1979, S. 1–9

[Ger03] Gerstenberger, Ralf: *ODEMx – Neue Lösungen für die Realisierung von C++-Bibliotheken zur Prozesssimulation*. Humboldt-Universität zu Berlin, Diplomarbeit, 2003. http://www2.informatik.hu-berlin.de/Institut/struktur/systemanalyse/zweckverband/gerstenberger2003.pdf

[GJBW09] Güneş, Mesut ; Juraschek, Felix ; Blywis, Bastian ; Watteroth, Olaf: DESCRIPT - A Domain Specific Language for Network Experiment Descriptions. In: *Inproceedings of the International Conference on Next Generation Wireless Systems (NGWS) 2009*. Melbourne, Australia, 12-14, October 2009

[GJS00] Gosling, James ; Joy, Bill ; Steele, Guy ; Gosling, James (Hrsg.): *The Java Language Specification*. Second Edition. Addison-Wesley Professional, 2000 (Java series). – 505 S. – ISBN 9780201310085

[GP01] Gehlsen, Björn ; Page, Bernd: A framework for distributed simulation optimization. In: B. A. Peters, D. J. M. J. S. Smith S. J. S. Smith (Hrsg.) ; Rohrer, M. W. (Hrsg.): *WSC '01: Proceedings of the 33nd conference on Winter simulation*. IEEE Computer Society, Washington, DC, USA, 2001, S. 508–514. – ISBN 0–7803–7309–X

[Gro10] Group, W3C Provenance I.: *What Is Provenance?* 2010. http://www.w3.org/2005/Incubator/prov/wiki/What_Is_Provenance. – abgerufen 05.03.2013

[GSB+08] Guzy, M. R. ; Smith, C. L. ; Bolte, J. P. ; Hulse, D. W. ; Gregory, S. V.: Policy Research Using Agent-Based Modeling to Assess Future Impacts of Urban Expansion into Farmlands and Forests. In: *Ecology and Society*, Bd. 13, Nr. 1, 2008. http://www.ecologyandsociety.org/vol13/iss1/art37/

[GSK+11] Görlach, Katharina ; Sonntag, Mirko ; Karastoyanova, Dimka ; Leymann, Frank ; Reiter, Michael : *Kapitel* 4 Conventional Workflow Technology for Scientific Simulation. In: Yang, Xiaoyu ; Wang, Lizhe ; Jie, Wei (Hrsg.): *Guide to e-Science*. 1. Edition. Springer-Verlag, 2011 (Computer Communications and Networks). – ISBN 978–0–85729–438–8, S. 1–31

[Hen95] Henriksen, James O.: An Introduction to SLX. In: *Proceedings of the 27th conference on Winter simulation*. IEEE Computer Society, Washington, DC, USA, 1995 (WSC '95), S. 502–509. – ISBN 0–7803–3018–8

[Hes09] Hessellund, Anders: *Domain-specific Multimodeling*. IT University of Copenhagen, Dissertation, 2009

[HMU79] Hopcroft, J. E. ; Motwani, R. ; Ullman, J. D.: *Introduction to Automata Theory, Languages, and Computation*. 3. Edition. Addison-Wesley, Boston, MA, USA, 1979. – ISBN 0321455363

[HR00] Harel, David ; Rumpe, Bernhard: Modeling Languages: Syntax, Semantics and All That Stuff, Part I: The Basic Stuff / Weizmann Science Press of Israel. Weizmann Science Press of Israel, Jerusalem, Israel, 08 2000. – Forschungsbericht

[HRFR06] Henderson, Thomas R. ; Roy, Sumit ; Floyd, Sally ; Riley, George F.: ns-3 project goals. In: *WNS2 '06: Proceeding from the 2006 workshop on ns-2: the IP network simulator*. ACM, New York, NY, USA, 2006, S. 13. – ISBN 1–59593–508–8

[HWL11] *Humboldt Wireless Lab (HWL)*. 05 2011. http://hwl.hu-berlin.de/. – abgerufen 05.03.2013

[HXTW09] Huang, B. ; Xie, C. ; Tay, R. ; Wu, B.: Land-use-change modeling using unbalanced support-vector machines. In: *Environment and Planning B: Planning and Design*, Bd. 36, Nr. 3, 2009, S. 398 – 416. http://dx.doi.org/10.1068/b33047. – DOI 10.1068/b33047

[JZH+12] Juraschek, Felix ; Zubow, Anatolij ; Hahm, Oliver ; Scheidgen, Markus ; Blywis, Bastian ; Sombrutzki, Robert ; Güneş, Mesut ; Fischer, Joachim: Towards Smart Berlin - an experimental facility for heterogeneous Smart City infrastructures. In: *Local Computer Networks Workshops (LCN Workshops), 2012 IEEE 37th Conference on* IEEE, 2012, S. 886–892

[Kar77] Karplus, Walter J.: The spectrum of mathematical modeling and systems simulation. In: *SIGSIM Simulation Digest*, Bd. 9, Nr. 1, September 1977, S. 32–38. http://dx.doi.org/10.1145/1102505.1102522. – DOI 10.1145/1102505.1102522. – ISSN 0163–6103. – ISSN 0163–6103

[KBA02] Kurtev, I. ; Bézivin, J. ; Aksit, M.: Technological spaces: An initial appraisal. In: *CoopIS, DOA'2002 Federated Conferences, Industrial track*, 2002

[KFA07] Kluth, Ronald ; Fischer, Joachim ; Ahrens, Klaus: *Ereignisorientierte Computersimulation mit ODEMx*. 11 2007. http://edoc.hu-berlin.de/docviews/abstract.php?id=29095 (Informatik-Berichte). – ISSN: 0863-095X

[Klu12] Kluge, Friedrich ; Seebold, Elmar (Hrsg.): *Etymologisches Wörterbuch der deutschen Sprache*. 24. Auflage. Walter de Gruyter, 2012. – 1112 S. – ISBN 9783111488608. – ISBN: 978-3110174731

[Knu64] Knuth, Donald E.: Backus Normal Form vs. Backus Naur Form. In: *Commun. ACM*, Bd. 7, 12 1964, S. 735–736. http://dx.doi.org/10.1145/355588.365140. – DOI 10.1145/355588.365140. – ISSN 0001–0782

[Knu90] Knuth, Donald E.: The Genesis of Attribute Grammars. 1990. http://dx.doi.org/10.1007/3-540-53101-7-1. In: Deransart, P. (Hrsg.) ; Jourdan, M. (Hrsg.): *Attribute Grammars and their Applications* Bd. 461. Springer Berlin / Heidelberg, 1990. – DOI 10.1007/3–540–53101–7–1. – ISBN 978–3–540–53101–2, S. 1–12

[Kre99] Kreplin, Klaus-Dieter: Konkordanz englischer und deutscher Begriffe des Workflow Management / Workflow Management Coalition (WfMC). 1999. http://www.wfmc.org/Download-document/Terminology-and-Glossary-German.html. 1999. – Forschungsbericht

[KS11] Kuchcinski, K. ; Szymanek, R.: *JaCoP – Java Constraint Programming solver*. 2011. http://www.jacop.eu/. – abgerufen 05.03.2013

[KTF09] Kühnlenz, Frank ; Theisselmann, Falko ; Fischer, Joachim: Model-driven Engineering for Transparent Environmental Modeling and Simulation. In: Troch, I. (Hrsg.) ; Breitenecker, F. (Hrsg.) ; Vienna University of Technology (Veranst.): *Proceedings MathMod Vienna 2009* Vienna University of Technology, 2009 (Argesim Report 35). – ISBN 978-3-901608-35-3

[Kun11] Kunert, Andreas: *Prozessorientierte optimistisch-parallele Simulation*. Humboldt-Universität zu Berlin, Mathematisch-Naturwissenschaftliche Fakultät II, Dissertation, 01 2011. http://edoc.hu-berlin.de/docviews/abstract.php?id=37551. – [Online: Stand 2011-02-03T16:29:10Z]

[KW78] Korn, Granino A. ; Wait, John V.: *Digital Continuous-system Simulation*. Prentice-Hall, 1978. – 320 S. – ISBN: 978-0132122740

[LAB+06] Ludäscher, Bertram ; Altintas, Ilkay ; Berkley, Chad ; Higgins, Dan ; Jaeger, Efrat ; Jones, Matthew ; Lee, Edward A. ; Tao, Jing ; Zhao, Yang: Scientific workflow management and the Kepler system. In: *Concurrency and Computation: Practice and Experience*, Bd. 18, Nr. 10, 2006, 1039–1065. http://dx.doi.org/10.1002/cpe.994. – DOI 10.1002/cpe.994

[LAB+09] Ludäscher, B. ; Altintas, I. ; Bowers, S. ; Cummings, J. ; Critchlow, T. ; Deelman, E. ; Roure, D. D. ; Freire, J. ; Goble, C. ; Jones, M. ; Klasky, S. ; McPhillips, T. ; Podhorszki, N. ; Silva, C. ; Taylor, I. ; Vouk, M.: *Kapitel* 13 Scientific Process Automation and Workflow Management. In: Shoshani, A. ; Rotem, D. (Hrsg.): *Scientific Data Management: Challenges, Existing Technology, and Deployment*. Chapman & Hall/CRC, 2009 (Computational Science Series). – ISBN 978-1420069808

[Lar04] Larman, Craig: *Applying UML and Patterns: An Introduction to Object-oriented Analysis and Design and Iterative Development*. 3. Edition. Prentice Hall, 2004. – 736 S. – ISBN 9780131489066

[Lee73] Lee, Douglass B.: Requiem for Large-Scale Models. In: *Journal of the American Institute of Planners*, Bd. 39, Nr. 3, Mai 1973, S. 163–178. http://dx.doi.org/10.1080/01944367308977851. – DOI 10.1080/01944367308977851. – ISSN 0002–8991

[LR00] Leymann, Frank ; Roller, Dieter: *Production Workflow: Concepts and Techniques*. Prentice Hall, 2000. – 480 S.

[LS11] Lumerical Solutions, Inc.: *FDTD Solutions*. 11 2011. http://www.lumerical.com/fdtd.php

[LST96] Lee, W. H. K. ; Shin, T. C. ; Teng, T. L.: Design and implementation of earthquake early warning systems in Taiwan. In: *Proc. 11th World Conference on Earthquake Engineering, Acapulco, Mexico*, Bd. Oxford, England: Pergamon, Disc 4, 1996, S. 2133

[LTK+12] Lakes, Tobia ; Theisselmann, Falko ; Kühnlenz, Frank ; Krüger, Carsten ; Fischer, Joachim: Modifying the SLEUTH model to simulate urban growth in Albania using language-centered model management. In: *Environmental Modelling & Software*, 2012. – In Review

[MAKP88] Moll, R.N. ; Arbib, M.A. ; Kfoury, A.J. ; Pustejovsky, J.: *An Introduction to formal Language Theory*. Springer Berlin, 1988 (Texts and monographs in computer science). – 203 S.

[MBZL09] McPhillips, T. ; Bowers, S. ; Zinn, D. ; Ludäscher, B.: Scientific workflow design for mere mortals. In: *Future Generation Computer Systems*, Bd. 25, Nr. 5, 2009, S. 541–551

[MC11] Mainzer, Klaus ; Chua, Leon O.: *The Universe as Automaton: From Simplicity and Symmetry to Complexity*. 1. Edition. Springer, 2011 (SpringerBriefs in Complexity). – 108 S. – ISBN 9783642234767

[MCD+05] Maechling, Philip ; Chalupsky, Hans ; Dougherty, Maureen ; Deelman, Ewa ; Gil, Yolanda ; Gullapalli, Sridhar ; Gupta, Vipin ; Kesselman, Carl ; Kim, Jihic ; Mehta, Gaurang ; Mendenhall, Brian ; Russ, Thomas ; Singh, Gurmeet ; Spraragen, Marc ; Staples, Garrick ; Vahi, Karan: Simplifying construction of complex workflows for non-expert users of the Southern California Earthquake Center Community Modeling Environment. In: *SIGMOD Rec.*, Bd. 34, Nr. 3, 2005, S. 24–30. http://dx.doi.org/10.1145/1084805.1084811. – DOI 10.1145/1084805.1084811. – ISSN 0163–5808

[MCF03] Mellor, Stephen J. ; Clark, Anthony N. ; Futagami, Takao: Guest Editors' Introduction: Model-Driven Development. In: *IEEE Software*, Bd. 20, Nr. 5, 09/10 2003, S. 14–18. http://dx.doi.org/10.1109/MS.2003.1231145. – DOI 10.1109/MS.2003.1231145. – ISSN 0740–7459

[Min65] Minsky, Marvin: Matter, Mind and Models. In: *Proceedings of International Federation of Information Processing Congress (IFIP) Congress 1965* Bd. 1, 1965, 45–49

[MM03] Miller, Joaquin ; Mukerji, Jishnu: MDA Guide Version 1.0.1 / OMG. 06 2003. http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf. 2003. – Forschungsbericht

[MM09] Milic, B. ; Malek, M.: NPART-node placement algorithm for realistic topologies in wireless multihop network simulation. In: *Proceedings of the 2nd International Conference on Simulation Tools and Techniques* ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009, S. 1–10

[Moo01] Moore, Reagan: Data Management Systems for Scientific Applications. In: *Proceedings of the IFIP TC2/WG2.5 Working Conference on the Architecture of Scientific Software*. Kluwer, B.V., Deventer, The Netherlands, The Netherlands, 2001, 273–284. – ISBN 0–7923–7339–1

[MWHW12] Missier, Paolo ; Woodman, S. ; Hiden, H. ; Watson, P.: Provenance and Data Differencing for Workflow Reproducibility Analysis / Newcastle upon Tyne: Newcastle University. 10 2012. http://www.cs.ncl.ac.uk/publications/trs/papers/1353.pdf. 2012 (School of Computing Science Technical Report Series 1353). – Forschungsbericht

[OAF+04] Oinn, T. ; Addis, M. ; Ferris, J. ; Marvin, D. ; Greenwood, M. ; Carver, T. ; Pocock, M.R. ; Wipat, A. ; Li, P.: Taverna: a tool for the composition and enactment of bioinformatics workflows. In: *Bioinformatics*, Bd. 20, Nr. 17, 05 2004, S. 3045 – 3054. – ISSN 1367–4803

[ORI+10] Oskooi, Ardavan F. ; Roundy, David ; Ibanescu, Mihai ; Bermel, Peter ; Joannopoulos, J. D. ; Johnson, Steven G.: MEEP: A flexi-

ble free-software package for electromagnetic simulations by the FDTD method. In: *Computer Physics Communications*, Bd. 181, Nr. 3, 01 2010, S. 687–702. http://dx.doi.org/10.1016/j.cpc.2009.11.008. – DOI 10.1016/j.cpc.2009.11.008

[PAM06] Pijanowski, B.C. ; Alexandridis, K.T. ; Mueller, D.: Modelling urbanization patterns in two diverse regions of the world. In: *Journal of Land Use Science*, Bd. 1, Nr. 2-4, 2006, S. 83–108. http://dx.doi.org/10.1080/17474230601058310. – DOI 10.1080/17474230601058310

[Par07] Parr, Terence: *The Definitive ANTLR Reference: Building Domain-Specific Languages*. First. Pragmatic Bookshelf, 2007 (Pragmatic Programmers). – ISBN 0978739256

[PCSF08] Pilato, C. ; Collins-Sussman, B. ; Fitzpatrick, B. ; Treseler, Mary E. (Hrsg.): *Version control with subversion*. O'Reilly Media, Incorporated, 2008. – 430 S. http://svnbook.red-bean.com/. – ISBN-13: 978-0596510336

[PLC00] Page, B. ; Lechler, T. ; Claassen, S.: *Objektorientierte Simulation in Java mit dem Framework DESMO-J-Java*. Books on Demand GmbH, 2000. – 197 S. – ISBN 9783831101634

[Pop05] Popper, Karl R. ; Keuth, Herbert (Hrsg.): *Logik der Forschung*. Bd. 3. 11. Auflage (Erstveröffentlichung 1935). Mohr Siebeck, 2005. – 601 S. – ISBN-13: 978-3161484100

[Sad09] Sadilek, Daniel A.: *Test-Driven Language Modeling*. Humboldt-Universität zu Berlin, Dissertation, 2009

[Sau06] Sauer, Robert: *Simulation standardbasierter Workflow-Modelle mit ODEMx*. Humboldt-Universität zu Berlin, Diplomarbeit, 2006

[SBPM09] Steinberg, David ; Budinsky, Frank ; Paternostro, Marcelo ; Merks, Ed ; Gamma, Erich (Hrsg.) ; Nackman, Lee (Hrsg.) ; Wiegand, John (Hrsg.): *EMF: Eclipse Modeling Framework*. 2. Revised Edition. Addison-Wesley Longman, Amsterdam, 2009. – 744 S. – ISBN 0321331885

[Sch74] Schriber, Thomas J.: *Simulation using GPSS*. Wiley, 1974. – 532 S. – ISBN 0471763101

[Sch06a] Schmidt, Douglas C.: Guest Editor's Introduction: Model-Driven Engineering. In: *Computer*, Bd. 39, Nr. 2, 02 2006, S. 25–31. http://dx.doi.org/10.1109/MC.2006.58. – DOI 10.1109/MC.2006.58. – ISSN 0018–9162

[Sch06b] Schmidt, Douglas C.: Model-Driven Engineering. In: *Computer*, Bd. 39, Nr. 2, 02 2006, S. 25–31. http://dx.doi.org/10.1109/MC.2006.58. – DOI 10.1109/MC.2006.58. – ISSN : 0018-9162

[Sch09] Scheidgen, Markus: *Description of Computer Languages Based on Object-Oriented Meta-Modelling*. Humboldt-Universität zu Berlin, Dissertation, 2009. http://www2.informatik.hu-berlin.de/Institut/struktur/systemanalyse/zweckverband/scheidgen-thesis-2009.pdf

[Sch10] Scheidgen, Markus: Textual Modelling Embedded into Graphical Modelling. In: *Model Driven Architecture – Foundations and Applications*, Bd. 5095/2010, 2010, 153–168. http://dx.doi.org/10.1007/978-3-540-69100-6. – DOI 10.1007/978–3–540–69100–6

[Sch11] Schmidt, Martin: *Einsatz modellgetriebener Entwicklung im Bereich der Experimentalphysik*. Beuth Hochschule für Technik Berlin, Diplomarbeit, 10 2011

[SF03] Smith, H. ; Fingar, P.: *Business process management: the third wave*. Bd. 1. Meghan-Kiffer Press, 2003. – 292 S. – ISBN 0-929652-33-9

[SGM02] Szyperski, Clemens ; Gruntz, Dominik ; Murer, Stephan: *Component Software: Beyond Object-oriented Programming*. Addison-Wesley, 2002

[SHF94] Schöneburg, Eberhard ; Heinzmann, Frank ; Feddersen, Sven: *Genetische Algorithmen und Evolutionsstrategien*. Addison-Wesley, 1994. – 481 S. – ISBN: 978-3893194933

[Sta73] Stachowiak, Herbert: *Allgemeine Modelltheorie*. Springer-Verlag, 1973. – 494 S. – ISBN 9783211811061

[sub11] *Apache Subversion*. 05 2011. http://subversion.apache.org/. – abgerufen 05.03.2013

[SV07] Stahl, Thomas ; Völter, Markus: *Modellgetriebene Softwareentwicklung: Techniken, Engineering, Management*. 2., aktualisierte und erweiterte Auflage. dpunkt Verlag, Heidelberg, 2007. – 458 S. http://nbn-resolving.de/urn:nbn:de:101:1-201211152584. ISSN 3–89864–310–7. – ISBN 3898644480

[SZFK12] Scheidgen, Markus ; Zubow, Anatolij ; Fischer, Joachim ; Kolbe, Thomas: Automated and Transparent Model Fragmentation for Persisting Large Models. In: *Model Driven Engineering Languages and Systems*, 2012, S. 102–118

[SZS12a] Scheidgen, Markus ; Zubow, Anatoli ; Sombrutzki, Robert: ClickWatch – An Experimentation Framework for Communication Network Test-beds. In: *Wireless Communications and Networking Conference (WCNC)*. Paris, 2012, S. 3296–3301

[SZS12b] Scheidgen, Markus ; Zubow, Anatolij ; Sombrutzki, Robert: HWL – A High-Performance Wireless Sensor Research Network / Humboldt-Universität zu Berlin. 2012. – Forschungsbericht

[TDF09] Theisselmann, Falko ; Dransch, Doris ; Fischer, Joachim: Model-driven Development of Environmental Modeling Languages: Language and Model Coupling. In: Wohlgemuth, B.; Voigt K. V.; Page P. V.; Page (Hrsg.): *EnviroInfo 2009 : Environmental Informatics and Industrial Environmental Protection: Concepts, Methods and Tools, 23rd International Conference on Informatics for Environmental Protection* Bd. 1: Conference Sessions, Shaker-Verlag, 09 2009, S. 409–417

[TH00] Taflove, Allen ; Hagness, Susan C.: *Computational Electrodynamics: The Finite-Difference Time-Domain Method*. 3. Artech House, 2000

[TH05] Torvalds, L. ; Hamano, J.: *GIT – fast version control system*. 2005. http://git-scm.com/. – abgerufen 05.03.2013

[TKK+10] Theisselmann, F. ; Kühnlenz, F. ; Krüger, C. ; Fischer, J. ; Lakes, T.: How to reuse and modify an existing land use change model? Exploring the benefits of language-centered tool support. In: Greve, Klaus (Hrsg.) ; Cremers, Armin B. (Hrsg.) ; Universität Bonn (Veranst.): *EnviroInfo 2010 : Integration of Environmental Information in Europe, 24th International Conference on Informatics for Environmental Protection* Universität Bonn, Shaker-Verlag, Aachen, 10 2010, S. 678 – 688. – ISBN 978-3-8322-9458-8

[Wac07] Wachsmuth, Guido: Metamodel Adaptation and Model Co-adaptation. 2007. http://dx.doi.org/10.1007/978-3-540-73589-2_28. In: Ernst, Erik (Hrsg.): *ECOOP 2007 – Object-Oriented Programming*. Springer, 2007 (Lecture Notes in Computer Science 4609), 600–624. – ISBN 978-3-540-73588-5

[Wan99] Wang, Rongjiang: A simple orthonormalization method for stable and efficient computation of Green's functions. In: *Bulletin of the Seismological Society of America*, Bd. 89, Nr. 3, 1999, 733-741. http://www.bssaonline.org/cgi/content/abstract/89/3/733

[WCI+07] Weber, E. ; Convertito, V. ; Iannaccone, G. ; Zollo, A. ; Bobbio, A. ; Cantore, L. ; Corciulo, M. ; Di Crosta, M. ; Elia, L. ; Martino, C. ; Romeo, A. ; Satriano, C.: An Advanced Seismic Network in the Southern Apennines (Italy) for Seismicity Investigations and Experimentation with Earthquake Early Warning. In: *Seismological Research Letters*, Bd. 78, Nr. 6, 2007, 622-634. http://dx.doi.org/10.1785/gssrl.78.6.622. – DOI 10.1785/gssrl.78.6.622

[WEK+09] Wenzel, F. ; Erdik, M. ; Köhler, N. ; Zschau, J. ; Milkereit, C. ; Picozzi, M. ; Fischer, J. ; Redlich, J.P. ; Kühnlenz, F. ; Lichtblau, B. ; Eveslage, I. ; Christ, I. ; Lessing, R. ; Kiehle, C.: EDIM – Earthquake Disaster Information System for the Marmara Region, Turkey. In: Stroink, Dr. L. (Hrsg.): *Early Warning Systems in Earth Management*. Koordinierungsbüro GEOTECHNOLOGIEN, Munich, 10 2009 (Geotechnologien Science Report 13), 84 - 95

[WfM99] ; Workflow Management Coalition (WfMC) (Veranst.): *Terminology and Glossary English, WFMC-TC-1011 Ver 3*. 02 1999. http://www.wfmc.org/Download-document/WFMC-TC-1011-Ver-3-Terminology-and-Glossary-English.html

[Wid11] Wider, Arif: Towards Lenses for View Synchronization in Metamodel-Based Domain-Specific Workbenches. In: *3rd Workshop Methodische Entwicklung von Modellierungswerkzeugen (MEMWe'11) at conference INFORMATIK*, 2011

[Wis11] Wissenschaftsrat: *Empfehlungen zu wissenschaftlichen Sammlungen als Forschungsinfrastrukturen*. 01 2011. http://www.wissenschaftsrat.de/download/archiv/10464-11.pdf

[Wis12] Wissenschaftsrat: *Empfehlungen zur Weiterentwicklung der wissenschaftlichen Informationsinfrastrukturen in Deutschland bis 2020*. 07 2012. http://www.wissenschaftsrat.de/download/archiv/2359-12.pdf

[WK06] Wimmer, Manuel ; Kramler, Gerhard: Bridging Grammarware and Modelware. In: Bruel, Jean-Michel (Hrsg.): *Satellite Events at the MoDELS 2005 Conference* Bd. 3844. Springer Berlin / Heidelberg, 2006. – ISBN 978–3–540–31780–7, S. 159–168

[WLC+00] Wu, Yih-Min ; Lee, William H. K. ; Chen, Chen-Chun ; Shin, Tzay-Chyn ; Teng, Ta-Liang ; Tsai, Yi-Ben: Performance of the Taiwan Rapid Earthquake Information Release System (RTD) during the 1999 Chi-Chi (Taiwan) Earthquake. In: *Seismological Research Letters*, Bd. 71, Nr. 3, 05 2000, 338-343. http://dx.doi.org/10.1785/gssrl.71.3.338. – DOI 10.1785/gssrl.71.3.338

[WSKF11] Wider, Arif ; Schmidt, Martin ; Kühnlenz, Frank ; Fischer, Joachim: A Model-Driven Workbench for Simulation-Based Development of Optical Nanostructures. In: Kocí, Radek (Hrsg.) ; Hanácek, Petr (Hrsg.) ; Kunovsky, Jirí (Hrsg.) ; Zboril, Frantisek (Hrsg.) ; Samek, Jan (Hrsg.) ; Peringer, Petr (Hrsg.): *Proceedings of the CSSim 2011 - Conference on Computer Modelling and Simulation*. Brno University of Technology (BUT), Brno, Czech Republic, 09 2011, 187 – 195. – ISBN 978-80-214-4320-4

[WWVM96] Wainer, J. ; Weske, M. ; Vossen, G. ; Medeiros, C.B.: Scientific workflow systems. In: *Proceedings of the NSF Workshop on Workflow and Process Automation in Information Systems: State of the Art and Future Directions*, 1996

[Xpa13] *M2T Xpand*. 02 2013. http://wiki.eclipse.org/Xpand. – abgerufen 05.03.2013

[YB05] Yu, J. ; Buyya, R.: A taxonomy of workflow management systems for grid computing. In: *Journal of Grid Computing*, Bd. 3, Nr. 3, 2005, S. 171–200

[Yee66] Yee, Kane: Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media. In: *Antennas and Propagation, IEEE Transactions on*, Bd. 14, Nr. 3, 05 1966, S. 302–307. http://dx.doi.org/10.1109/TAP.1966.1138693. – DOI 10.1109/TAP.1966.1138693

[YGN09] Yildiz, Ustun ; Guabtni, Adnene ; Ngu, Anne H.: Business versus Scientific Workflows: A Comparative Study. In: *Services, IEEE Congress on*, Bd. 0, 07 2009, S. 340–343. http://dx.doi.org/10.1109/SERVICES-I.2009.60. – DOI 10.1109/SERVICES–I.2009.60. ISBN 978–0–7695–3708–5

[Yu12] Yu, Jingyang: *Provenance in a Metamodel-based Workflow Management System*. Humboldt-Universität zu Berlin, Diplomarbeit, 07 2012

[Z.102] ; ITU-T Z.100 (Veranst.): *Specification and Description Language (SDL)*. 2002

[ZPK00] Zeigler, Bernhard P. ; Praehofer, Herbert ; Kim, Tag G.: *Theory of Modeling and Simulation*. Second Edition. Academic press, San Diego, California 92101-4495, USA, 2000. – 510 S. – ISBN: 0-12-778455-1

Abkürzungsverzeichnis

AKID	Atomarität, Konsistenz, Isoliertheit und Dauerhaftigkeit (im englischen ACID) . . . 93
ANTLR	ANother Tool for Language Recognition (siehe [Par07]) . . . 123
ASN.1	Abstact Syntax Notation One (siehe [Dat98]) . . . 82
BMBF	Bundesministerium für Bildung und Forschung . . . 80
BNF	Backus-Naur-Form . . . 65
BPEL	WS-Business Process Execution Language . . . 12
BPM	Business-Process-Modeling . . . 39
BPMN	Business Process Model and Notation . . . 57
B-Wf	Business-Workflow . . . 39
CDO	Connected Data Objects . . . 124
CMOF	Complete Meta Object Facility (siehe [20006a]) . . . 121
CORBA	Common Object Request Broker Architecture . . . 136
CPU	Central-Processing-Unit . . . 95
CSP	Constraint-Satisfaction-Problem (dt.: Bedingungserfüllungsproblem) . . . 116
DAG	Directed-Acyclic-Graph . . . 51
DCG	Directed-Cyclic-Graph . . . 51
DES	Distributed Embedded System . . . 11
DESMO-J	Discrete Event Simulation and MOdelling in Java . . . 38
DESS	Differential-Equation-System-Specification . . . 33
DES-TBMS	DES-Testbed Management System . . . 11
DEVS	Discrete-Event-System-Specification . . . 33
DFG	Deutsche Forschungsgemeinschaft . . . 235
DISMO	Distributed-Simulation-Optimisation (siehe [GP01]) . . . 38
DSL	Domain-Specific-Language . . . 59
ECAL	Environmental-Cellular-Automata-Language . . . 74
EDIM	Earthquake Disaster Information System for the Marmara Region, Turkey (siehe [WEK+09]) . . . 80
EEWS	Earthquake Early Warning System . . . 82
EMF	Eclipse Modeling Framework (siehe [SBPM09]) . . . 121
EMOF	Essential Meta Object Facility . . . 121
EMS	Experiment-Management-System (für SOSEWIN, siehe [FKAE09b]) . . . 81
EU	Europäische Union . . . 80
Exp-Wf	Experimentier-Workflow . . . 42
ExpL	Experimentation-Language . . . 3
ExpL-Wf	ExpL-Workflow . . . 101

ExpL-Wfe ExpL-Workflow-Engine 128
ExpL-WfMS ExpL-Workflow-Management-System 101
FDM Finite-Differenzen-Methode (siehe [TH00]) 78
FDTD finite-difference time-domain (*englische Schreibweise*) 78
GFZ Helmholtz-Zentrum Potsdam, Deutsches GeoForschungsZentrum 80
GIS Geoinformationssystem 87
GPL General-Purpose-Language 59
GPSS General Purpose Simulation System (siehe [Sch74]) 3
GUI Graphical-User-Interface 79
HUB Humboldt-Universität zu Berlin 80
HWL Humboldt-Wireless-Lab (siehe [SZS12b, HWL11]) 36
IEEE Institute of Electrical and Electronics Engineers 92
JiST Java in Simulation Time 36
KOERI Kandilli Observatory and Earthquake Research Institute 80
LCA Language-Centered-Approach 101
M&S Modellierung & Simulation 5
M2C Modell-zu-Code(-Transformation) 120
M2M Modell-zu-Modell(-Transformation) 120
M2T Modell-zu-Text(-Transformation) 121
MDA Model-Driven-Architecture (siehe [MM03]) 26
MDD Model-Driven-Development (siehe [MCF03]) 119
MDSD Model-Driven-Software-Development (siehe [SV07]) 119
MDE Model-Driven-Engineering (siehe [Sch06a]) 9
METRIK Modellbasierte Entwicklung von Technologien für selbstorganisierende dezentrale Informationssysteme im Katastrophenmanagement 5
MOF Meta Object Facility (siehe [20006a]) 67
MUX Model-Under-Experimentation (bezogen auf ein computerbasiertes Modell) 23
MVC Model-View-Controller 104
MWE Modeling Workflow Engine 125
ns-3 network simulator 3 (siehe [HRFR06]) 84
oAW openArchitectureWare 122
OCL Object Constraint Language 123
ODEM Object-oriented-Discrete-Event-Modelling (siehe [FA96]) 35
ODEMx ODEM mit Erweiterung (siehe [Ger03]) 35
OMG Object Management Group (siehe [6]) 26
OOMM Objektorientierte Meta-Modellierung 66
OSGi Open Services Gateway initiative (frühe Bezeichnung) 136
PI Persistenter-Identifier 105
PIM Platform Independent Model 26
PSM Platform Specific Model 26
RDBMS relationales Datenbankmanagementsystem 124
RMI Remote Method Invocation 38
RTDS Real Time Developer Studio (siehe [20009f]) 34

SAFER Seismic eArly warning For EuRope (siehe [20006b]) 80
SDL Specification- and Description-Language (siehe [Z.102]) 84
SDL-RT SDL-Real-Time 82
SHK studentische Hilfskraft 77
SI Système international d'unités (Internationales Einheitensystem) 109
SIMPAS Simulation of Processes and Systems (siehe [Bry80]) 31
SLEUTH Slope, Land-cover, Exclusion, Urbanization, Transportation, Hillshade . . . 72
SLEUTH* Re-Implementierung von SLEUTH mit zusätzlichen Statistikwerten 74
SLEUTH*P SLEUTH* mit Populationstreiber 74
SLEUTH*(P) Modellklassen SLEUTH* und SLEUTH*P 74
SLEUTH(*P) Modellklassen SLEUTH, SLEUTH* und SLEUTH*P 74
SLX Simulation Language with Extensibility (siehe [Hen95]) 37
SOSEWIN Self-Organizing Seismic Early Warning Information Network (siehe [FPM+09]) 80
SSH Secure Shell 135
SVM Support-Vector-Machine 71
SWANS Scalable Wireless Ad-hoc Network Simulator 36
S-Wf Scientific-Workflow 41
TEF Textual Editing Framework (siehe [Sch10]) 123
TMF Textual Modeling Framework 122
uDigEE uDig Erdbeben-Edition (siehe [Eve08]) 85
UGM Urban Growth Model 73
UML Unified Modeling Language (siehe [20007b]) 65
VM Virtuelle Maschine 84
Wfe Workflow-Engine 48
WfMC Workflow Management Coalition 39
WfMS Workflow-Management-System 40
WLAN Wireless LAN 84
XML Extensible Markup Language 66
XPDL XML Process Definition Language (Standard der WfMC) 12

Glossar

Aktivität Beschreibung einer logischen Arbeitseinheit in einer Prozessdefinition. Besteht aus Elementaraufgaben. Synonym zu *Task*. 40

Applikationsdaten Sind anwendungsspezifisch und nicht von einem Workflow-Management-System zugreifbar. 50

Artefakt (Software) Allg. eine Datei-Ressource in einem Repositorium bzw. speziell eine bedeutungstragende Struktur für eine ExpL-Wf-Aktivität (z. B. Variablendeklaration, Variablenbelegung). 106

Automatisierbarkeit Automatisierbarkeit durch ein Computersystem bedeutet, dass der zugrundeliegende Experimentier-Prozess nur aus automatisierten Aktivitäten besteht, die durch ein Computersystem ausgeführt werden können. 90

Computersprachen Sind alle Sprachen, mit deren Hilfe Informationen an einen Computer oder zwischen Computern übermittelt werden. 59

Domäne Im Kontext von DSLs sowohl eine Wissensdomäne (Wissensgebiet, das definiert ist durch die Gesamtheit des Wissens innerhalb eines Fachbereichs) als auch eine Anwendungsdomäne (abgrenzbares Problemfeld oder einen bestimmten Einsatzbereich für Computersysteme oder Software). 59

Experiment Bezeichnet allgemein eine methodisch angelegte, zielgerichtete Untersuchung eines Systems zur empirischen Gewinnung von Daten über das System mit festgelegtem Werten für die Eingangsgrößen des Systems. In dieser Arbeit wird stellvertretend für das System ein computerbasiertes Modell des Systems untersucht. 20

Experimentier-Workflow Ist ein spezifisch strukturierter Scientific-Workflow, der aus den geordneten Phasen Planung, Ausführung und Auswertung besteht. Er ist von einer konkreten Notationsform unabhängig. 42

Experimentlauf Das Experiment und die durch den entsprechenden Versuch gewonnenen, empirischen Daten. 19

Experimentserie Menge von Experimenten, die durch einen Experimentplan beschrieben sind. 19

ExpL-Kern Bezeichnet eine DSL zur Beschreibung der Domäne *Experimentieren mit computerbasierten Modellen*. Bestandteil von ExpL. 101

ExpL Bezeichnet eine Sprachbeschreibung, die durch Sprachkopplung verschiedener DSLs entstandt und die Experimentier-Workflows der Experimentier-Domäne eines Fachbereichs (MUX-Domäne) beschreibt. ExpL besteht aus: ExpL-Kern, Konfigurations-DSL, Ausführungs-DSL und Auswertungs-DSL. 102

Flexibilität Teilbeschreibungen des Experimentier-Prozesses bilden komponierbare Einheiten, die unverändert wiederverwendbar und in ihrer Reihenfolge austauschbar sind. 91

Hypozentrum Georäumlicher Punkt unter der Erdoberfläche, an dem die Bruchfläche eines Erdbebens ihren Ursprung hat. 81

Metametamodell *Modell eines Metamodells*: Beschreibt die Struktur einer Menge von konformen Metamodellen mit Hilfe von Klassen und Relationen zwischen Klassen. Eine oft verwendete Analogie ist die eines „Werkzeugkastens“. 68

Metamodell *Modell eines Modells*: Beschreibt die Struktur einer Menge von konformen Modellen mit Hilfe von Klassen und Relationen zwischen Klassen. 66

Modell Verkürzte Abbildung (bzw. Abstraktion/Repräsentation) eines Systems, um ein bestimmtes Untersuchungsziel erreichbar zu machen. Hier synonym verwendet zu computerbasiertem, mathematischem Modell. 21

Moore-Nachbarschaft Die Moore-Nachbarschaft (auch 8er-Nachbarschaft bezeichnet) ist eine nach Edward F. Moore benannte Nachbarschaftsbeziehung in einem quadratischen Raster. Alle Flächen, welche mindestens eine Ecke mit der Basisfläche gemeinsam haben, gelten als Nachbarn. 72

MUX-Domäne Bezeichnet die Domäne (oder einen ihrer Teilbereiche), zu der das System gehört, welches durch das *Model-Under-Experimentation (MUX)* beschrieben ist. 23

MUX-Programm Ist ein ausführbares Programm, das aus dem *Model-Under-Experimentation (MUX)* erzeugt wurde. In einem MUX-Programm wird ein initialer Modellzustand des MUX hergestellt und dessen Verhaltensmethoden angesteuert. 24

MUX Das *Model-Under-Experimentation (MUX)* ist ein Simulationsmodell ohne Experiment-spezifische Anteile. Es ist Gegenstand von computerbasierten Modellexperimenten. 23

Nachnutzbarkeit Die Nachnutzbarkeit eines Experimentier-Prozesses bedeutet, dass er in Teilen oder als Ganzes erneut in einem anderen Experimentier-Prozess wiederverwendet werden kann. 91

Photonischer Kristall Photonische Kristalle sind periodische, optische Nanostrukturen mit einer Wellenlänge kleiner der des sichtbaren Lichtes. Sie beeinflussen die Bewegung von Photonen ähnlich, wie Heilbleiter die Bewegung von Elektronen beeinflussen (*optische Halbleiter*). 77

Prozessdefinition Definiert einen realweltlichen Prozess und besteht aus Aktivitäten. 40

Prozess Im Workflow-Kontext: Zeitlich und räumlich strukturierte Reihenfolge von realweltlich ablaufenden Schritten mit spezifischem Ziel. . 39

Reproduzierbarkeit bezeichnet die Möglichkeit, das gleiche Ergebnis eines Experimentes zu reproduzieren. Dazu ist eine Beschreibung des Experimentes, dessen Ergebnis und der zugrundeliegenden Daten notwendig. 90

Ressource Ist ein Betriebsmittel, um eine Handlung zu tätigen oder einen Vorgang ablaufen zu lassen. Beispiele: Ein Mensch oder eine Maschine. 40

Scientific-Workflow Ist eine formale Prozessdefinition zum Erreichen eines wissenschaftlichen Zieles mit Ausrichtung auf Daten und Ressourcen. 41

Simulation Ist ein Experiment, das auf Basis eines computerbasierten Modells mit Hilfe eines Simulators ausgeführt wird. 22

Simulator Ist ein Computersystem, welches ermöglicht, das Verhalten eines Modells auszuführen. Unter einem Simulator wird allgemein die Einheit von Simulationsprogramm (bzw. hier MUX-Programm), Betriebssystem und Hardware verstanden. 24

Sprachwerkzeuge Alle (Software-)Werkzeuge, die zur Definition von Syntax, Notation und Semantik einer Sprachinstanz und ihrer Verwendung dienen. 65

System Ist ein Pänomen, das abgrenzbar, kontrollierbar und beobachtbar ist. Ein System hat einen erkennbaren Systemzweck. Es akzeptiert Eingaben und erzeugt Ausgaben, zwischen denen ein ausschließlicher, kausaler Zusammenhang besteht. 20

Technologieraum Beschreibt einen Kontext mit einer Menge von zugehörigen Konzepten, einer Wissensbasis, Werkzeugen, notwendigen Fähigkeiten und Möglichkeiten. 65

Transparenz Abstrakter Oberbegriff für *Verständlichkeit*, *Nachnutzbarkeit* und *Reproduzierbarkeit* und fasst die grundlegenden Anforderungen an wissenschaftliches Arbeiten auf Basis des Experimentierens mit computerbasierten Modellen zusammen. 92

Wiederholbarkeit Strengere Reproduzierbarkeit, bei der das *exakt gleiche* Ergebnis erreicht werden muss. 90

Workflow-Applikation Ist eine Bezeichnung für ein Software-Programm, das mit einem Workflow-Enactment-Service interagiert, um Teile der Ausführung von Aktivitäten durchzuführen. 49

Workflow-Definition Eine Prozessdefinition, die ausschließlich aus automatisierten Aktivitäten komponiert ist. Synonym zu Workflow. 40

Workflow-Engine Ist ein Software-Dienst, der die Laufzeit-Ausführungsumgebung für eine Workflow-Instanz bereitstellt. 48

Workflow-Kontrolldaten Daten, die den dynamischen Zustand des Workflow-Management-Systems und seiner Workflow-Instanzen repräsentieren. Werden vom Workflow-Management-System und/oder einer Workflow-Engine gemanagt. 49

Workflow-Lebenszyklus Besteht aus Aktionen, die für Definition, Instanziierung und Ausführung von Workflow-Instanzen nötig sind. 40

Workflow-Management-System Ermöglicht die Definition, die Erzeugung und das Management der Ausführung von Workflows durch den Einsatz von Software, die auf einer oder mehrerer Workflow-Engines basiert. 47

Workflow-relevante Daten Daten, die den Zustandsübergang einer Workflow-Instanz charakterisieren. 50

Workflow-System Bezeichnet die Gesamtheit von Workflow-Definitionen, zugehörigen Instanzen und dem Workflow-Management-System, das sie verwaltet. 47

Zellulärer Automat Nutzbar zur Modellierung räumlich diskreter dynamischer Systeme. Die Zustandsübergänge einzelner Zellen zum Zeitpunkt t+1 hängen primär von den Zellzuständen in einer vorgegebenen Nachbarschaft und vom eigenen Zustand zum Zeitpunkt t ab. 72

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Danksagungen

Mein erster Dank gilt meinem Betreuer, Prof. Dr. Joachim Fischer, der mich gefördert und gefordert hat in allen Stadien meines Promotionsvorhabens. Seine stets fundierten Fragen und kritischen Anmerkungen erwiesen sich als sehr wertvoll und hilfreich.

Mein Dank gilt zudem meinen ehemaligen Kollegen vom Lehrstuhl *Systemanalyse*, vor allem Ingmar Eveslage, Dr. Klaus Ahrens und Manfred Hagen. Sie waren insbesondere auch in den Projekten SAFER und EDIM kompetente, kollegiale und team-orientierte Mitarbeiter. Das trifft ebenfalls auf meine ehemaligen Kollegen in diesen Projekten vom Nachbarlehrstuhl *Systemarchitektur* zu: Björn Lichtblau und Jens Nachtigall.

Während meiner Promotionszeit war ich Mitglied des Graduiertenkollegs METRIK, das von der DFG gefördert wurde (GRK 1324). Ich möchte mich bei den METRIK-Professoren bedanken für ihre Geduld und die ebenso hilfreichen, wie auch nicht selten kritischen Anmerkungen zu meiner Arbeit. Mit einer Metapher lässt es sich so ausdrücken: Nur durch Reibung entsteht Hitze, die man nutzen kann, um etwas schmackhaftes zu kochen.

Mit vielen der METRIKer aus der ersten und zweiten Generation habe ich interessante Gespräche geführt und viele von ihnen haben mich inspiriert und unterstützt. Ich kann nicht alle namentlich nennen, jedoch möchte ich mich besonders bedanken bei: Dr. Markus Scheidgen und Dr. Daniel Sadilek. Ebenso Siamak Haschemi, der mir stets hilfreich zur Seite stand bei technologischen Problemen und auch bei den Tücken einzelner Werkzeuge. Für eine sehr gute, unmittelbare Zusammenarbeit bei den Fallstudien meiner Arbeit und ebenso für viele anregende Gespräche bedanke ich mich bei Falko Theisselmann, Arif Wider, Martin Schmidt und Carsten Krüger. Ohne ihr Engagement wäre mancher Kontakt zu anderen Disziplinen nicht möglich gewesen.

Meinem langjährigen Freund Dr. Andreas Kunert bin ich zu besonderem Dank verpflichtet. Bereits als Kollege am Lehrstuhl *Systemanalyse* war er ein äußerst kompetenter, hilfsbereiter und scharfsinniger Begleiter. Diese Eigenschaften machten ihn auch zu einem idealen Lektor, wobei ich ihm für die Übernahme dieser Aufgabe besonders danke. Umso erfreulicher ist es, dass ich ihn auch zu meinen aktuellen Kollegen am *Computer- und Medienservice* der Humboldt-Universität zu Berlin zählen kann, bei denen ich mich ebenfalls bedanken möchte. Insbesondere danke ich meinem Abteilungsleiter Daniel Rohde für sein Verständnis, welches mir erlaubte, der Fertigstellung meiner Dissertationsschrift eine so hohe Priorität einräumen zu können. Meinem langjährigen und besten Freund Dennis Reinert möchte ich ebenfalls sehr herzlichen Dank für seine moralische Unterstützung und das Durchsehen meiner Arbeit übermitteln.

Als erster und wichtigster Person, die nicht aus meinem akademischen Umfeld stammt, möchte ich mich bei meiner Lebensgefährtin Zuzanna bedanken. Ihr gilt größter Dank für ihr

Verständnis und dass sie mir gerade in der Endphase des Niederschreibens viel Unterstützung zukommen ließ. Großer Dank gebührt auch meiner Mutter, die als erfahrene Autorin von Romanen und Kurzgeschichten meine Arbeit durchgesehen hat. Dabei wurde sie unterstützt von meinem Papa, dem ich hiermit auch herzlichst danken möchte.

Zuletzt möchte ich all den (anonymen) Personen danken, welche die Werkzeuge für den Inhalt und das typographische Setzen dieser Arbeit bereit gestellt haben. Dabei kann ich leider nur eine kleine Auswahl nennen – mein Dank geht an:

- die Entwickler von Eclipse (inkl. EMF), Xtext, Xpand, Xtend, CDO und MWE,
- die Entwickler des Werkzeugs PlantUML, mit dem viele Diagramme in dieser Arbeit mittels einer textuellen Beschreibung erstellt werden konnten,
- die Entwickler von LyX, LaTeX und von LuaLaTeX, die somit die typographische Form dieser Arbeit ermöglicht haben,
- die Designer des Fonts „Garamond Pro“ (für Fließtext) der Firma Adobe,
- die Designer des Fonts „`Source Code`“ (`Listings` einer Programmiersprache, Auszüge aus `Log-Dateien`, etc.) und
- die Designer des Fonts „FF Scala Sans Pro“ (Text in Abbildungen), namentlich Martin Majoor.

Selbständigkeitserklärung

Hiermit erkläre ich, dass

- ich die vorliegende Dissertationsschrift
  »Design und Management von Experimentier-Workflows«
  selbständig und ohne unerlaubte Hilfe angefertigt habe,
- ich mich nicht bereits anderwärts um einen Doktorgrad beworben habe oder einen solchen besitze und
- mir die Promotionsordnung der Mathematisch-Naturwissenschaftlichen Fakultät II der Humboldt-Universität zu Berlin bekannt ist gemäß des Amtlichen Mitteilungsblattes Nr. 34/2006.

Berlin, den 27.03.2013 Unterschrift: Frank Kühnlenz